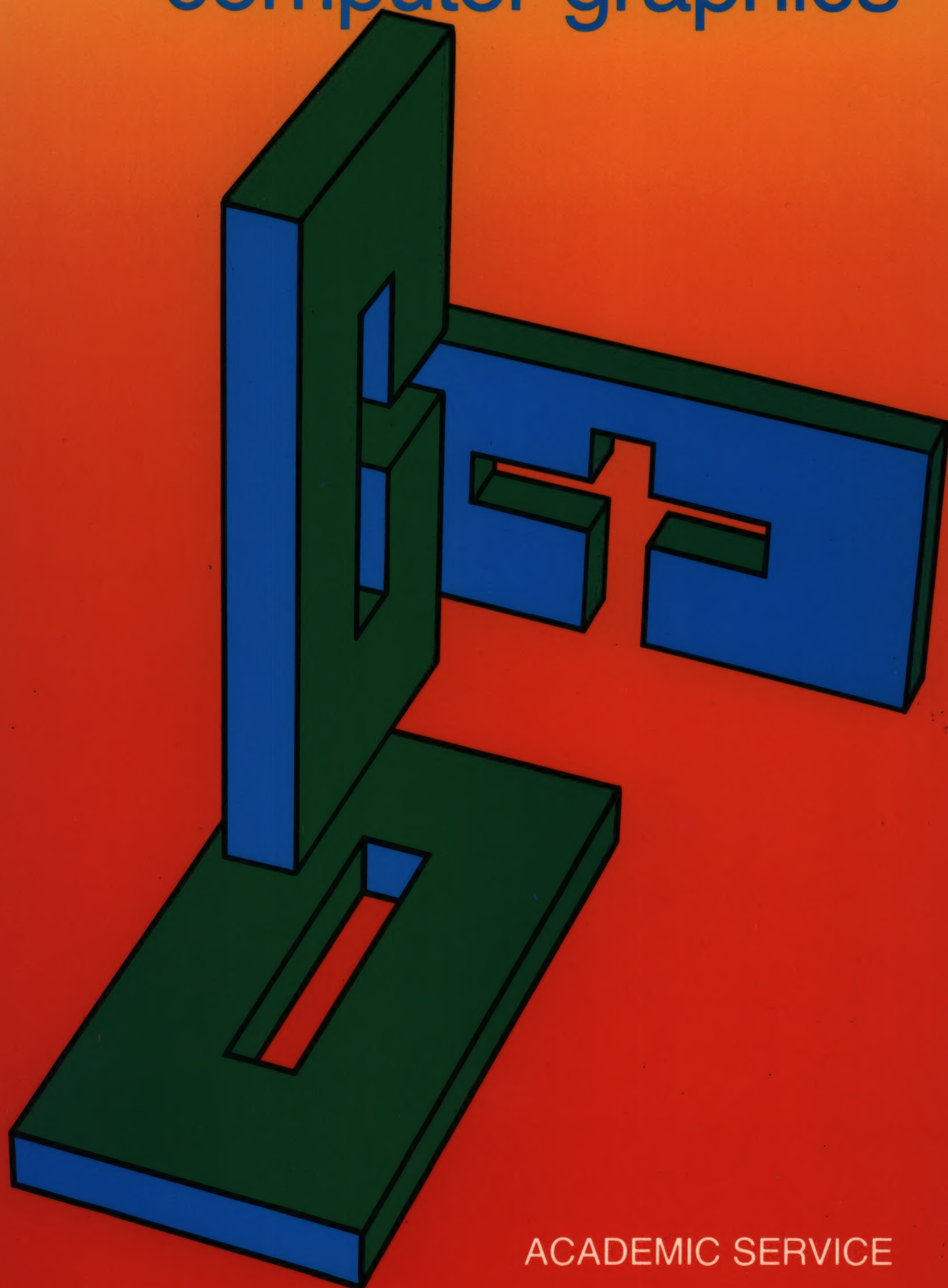
Leen Ammeraal

Interactieve 3D computer graphics



ACADEMIC SERVICE

INTERACTIEVE
3D COMPUTER GRAPHICS

Van dezelfde auteur zijn verschenen:

De programmeertaal C
Interactieve 3D computer graphics
Turbo C

Andere boeken over C:

Handboek voor de C-programmeur – Bolsky
C voor gevorderden – Sobelman & Krekelberg

Leen Ammeraal

Interactieve 3D computer graphics

ACADEMIC SERVICE

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Ammeraal, L.

Interactieve 3D computer graphics / L. Ammeraal. -
Schoonhoven : Academic Service. - Ill.
Met 2 diskettes. - ISBN 90-6233-387-7
ISBN 90-6233-386-9
SISO 527.5 SVS 8.12.3 UDC 681.3:76 NUGI 856
Trefw.: computergraphics.

10 9 8 7 6 5 4 3 2 1

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven

Zetwerk: RCO Amsterdam
Omslag: Prographics, Bussum
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam

ISBN 90 6233 386 9
NUGI 856

INHOUD

WOORD VOORAF

Net als de computer zelf, kan 'Computer Graphics' bezien worden vanuit twee verschillende gezichtspunten. Er zijn *gebruikers*, die vooral kijken naar de uiterlijke aspecten van grafische software en er zijn anderen die meer willen weten van het inwendige ervan, bijvoorbeeld omdat zij van plan zijn zelf soortgelijke programmatuur te schrijven. Laten we deze laatsten kortheidshalve als *programmeur* betitelen, hoewel ik mij ervan bewust ben dat mensen die programma's schrijven zichzelf meestal anders noemen. In tegenstelling tot de vorige door mij geschreven boeken, is dit boek bedoeld voor beide groepen lezers: de hoofdstukken 1 en 2 zijn vooral geschreven voor gebruikers, de hoofdstukken 3 en 4 voor programmeurs. Dit betekent overigens niet dat zij die alleen in de algoritmen en de programmatekst geïnteresseerd zijn de hoofdstukken 1 en 2 kunnen overslaan: voordat we ons verdiepen in de werking van een programma dienen we eerst te weten wat het geacht wordt te doen. Omgekeerd zouden echte gebruikers, die mogelijk totaal niet geïnteresseerd zijn in de programmatekst, het misschien toch op prijs kunnen stellen dat alle besproken programma's in het boek zijn opgenomen. Het komt dikwijls voor dat gebruikers enthousiast zijn over hun programmatuur, met uitzondering van één of twee punten, die zij verbeterd zouden willen zien. Als u de programma's in dit boek gewijzigd wilt hebben, dan bent u niet afhankelijk van de auteur (hoewel ik graag van eventuele problemen op de hoogte gesteld zou willen worden), maar dan kunt u uw wensen ook voorleggen aan iedere C-programmeur in wie u voldoende vertrouwen heeft. Complexe software is, net als een keten, even sterk als zijn zwakste schakel en hoe meer middelen u heeft om eventuele zwakke schakels te vervangen, des te beter. De laatste opmerking heeft niet de bedoeling te impliceren dat ik moeilijkheden verwacht. Ik heb het interactieve programma D3D, dat het hoofdonderwerp van dit boek vormt, gebruikt voor het tekenen van een groot aantal driedimensionale voorwerpen zonder daarbij op problemen te stuiten. Om snel een beeld van de mogelijkheden van D3D te krijgen zou u alvast een blik kunnen werpen op de illustraties die voorkomen aan het eind van elk van de hoofdstukken 1, 2 en 3.

In tegenstelling tot sommige meer algemene boeken over interactieve computer graphics en CAD is dit boek erg specifiek ten aanzien van de te gebruiken apparatuur en basisprogrammatuur. U dient te beschikken over een IBM PC (XT of AT) of een IBM PS/2 (of iets wat hiermee compatibel is); uiteraard dient ook een

grafische adapter (CGA, EGA, VGA, of HGA) aanwezig te zijn. Alle programma's in dit boek zijn geschreven in de taal C; onder de vele goede C-compilers die voor de IBM-PC beschikbaar zijn heb ik deze keer Turbo C van Borland gebruikt, een compiler die bijzonder goed en goedkoop is. Let wel, als u slechts een CAD-*gebruiker* bent en niet bekend bent met compilers en dergelijke zaken, dan kunt u eenvoudig de diskette die bij dit boek behoort bestellen bij de uitgever. Behalve programmatekst (source code), bevat deze diskette ook direct uitvoerbare programma's (xxx.EXE-files), zodat u er direct mee aan de slag kunt. Dit bespaart u het compileren en 'linken'; de programma's hoeven zelfs niet te worden 'geïnstalleerd' (dat wil zeggen aangepast aan uw configuratie), omdat zij zelf uitzoeken welke grafische adapter u in gebruik heeft.

In onderwijssituaties waar de grafische uitvoer van een IBM PC direct aan leerlingen of studenten kan worden getoond, bijvoorbeeld door gebruik te maken van een overhead projector, zou programma D3D nuttig kunnen zijn bij het onderwijs in de wiskunde en in vele andere vakken. Zowel voor studenten als voor docenten kan D3D goede diensten bewijzen in gevallen waarin het ruimtelijk voorstellingsvermogen te kort schiet. Verder is het maken van aardige plaatjes met de computer gewoon leuk: als studenten dit in het kader van hun studie moeten kunnen doen kan hun motivatie er wellicht positief door beïnvloed worden.

Hoewel ik zojuist heb gepoogd gebruikers van een IBM-PC ervan te overtuigen dat dit boek precies datgene is wat zij nodig hebben, wil ik ook benadrukken dat veel van de behandelde stof machine-onafhankelijk is. Er zullen nogal wiskundig gerichte onderwerpen aan de orde komen, zoals bijvoorbeeld driedimensionale rotaties, ruimtelijke krommen, Platonische lichamen, verschillende methoden om een bol te benaderen, B-splineoppervlakken en de eliminatie van verborgen lijnen. U kunt daarom de IBM PC beschouwen als niet meer dan een voorbeeld van een stuk gereedschap dat wiskundige begrippen nuttig maakt voor de praktijk.

De programmeertaal C staat bekend om zijn portabiliteit en Turbo C ondersteunt zowel de klassieke stijl van Kernighan & Ritchie als de moderne voorgestelde ANSI-standaard; de voornaamste verschillen tussen deze twee stijlen worden in detail besproken in paragraaf 3.1, dat wil zeggen in de eerste paragraaf die bedoeld is voor programmeurs.
Er is een relatie tussen dit boek en mijn vorige boeken *Programming Principles in Computer Graphics* (*PPCG*) en *Computer Graphics for the IBM PC* (*CGIP*). Perspectivische representaties van driedimensionale voorwerpen en eliminatie van verborgen lijnen worden behandeld in *PPCG* en laag-niveau grafische functies in *CGIP*. In het onderhavige boek worden deze belangrijke onderwerpen voornamelijk behandeld als stukken gereedschap, gebruikt in programma D3D. Ook in dit boek wordt de verborgen-lijn-algoritme (die nu efficiënter en algemener geïmplementeerd is) kort uiteengezet, maar *PPCG* zal geschikter zijn om het

wezenlijke van deze algoritme te bestuderen. Analoog hieraan, worden de laag-niveau grafische routines, die hier opgenomen zijn in paragraaf 4.3, in *CGIP* meer gedetailleerd besproken. Een ander verschilpunt met *CGIP* is dat het onderhavige boek ook enige aandacht besteedt aan de nieuwe grafische faciliteiten van Turbo C, die pas in Versie 1.5 beschikbaar gekomen zijn; hiervan wordt gebruik gemaakt in een alternatief grafisch pakket, eveneens opgenomen in paragraaf 4.3. Bovendien is het nu mogelijk grafische uitvoer op een plotter te verkrijgen, zoals we zullen zien in paragraaf 4.5. Veel illustraties in dit boek zijn op die manier tot stand gekomen, dank zij het feit dat ik gebruik kon maken van een Hewlett-Packard 7475A plotter, gekoppeld aan een IBM PC/AT, aan de Hogeschool Utrecht, afdeling Werktuigbouwkunde, lokatie Hilversum.

Het is misschien goed nog enkele woorden te wijden aan de Nederlandse vertaling, zowel van dit boek zelf als van de besproken programma's. Evenals in de Engelse versie heb ik getracht mij te houden aan de stelregel 'Schrijf zoals je spreekt'. Ik heb er daarom van afgezien, vaktermen zoals bijvoorbeeld 'exploded view' en 'printer' door zuiver Nederlandse (maar ongebruikelijke) woorden te vervangen. In de programmatekst zijn namen van variabelen niet vernederlandst, commentaarregels wel. Belangrijker is de tekst die de gebruiker op het beeldscherm ziet. Deze is bijna steeds in het Nederlands; een uitzondering is gemaakt voor menucommando's die tot één letter worden afgekort. Vertaling van woorden als *Clear* (*C*) en *Step* (*S*) in Nederlandse woorden met een eenduidige eerste letter zou voor de gebruiker meer last dan gemak hebben opgeleverd en is daarom achterwege gebleven.

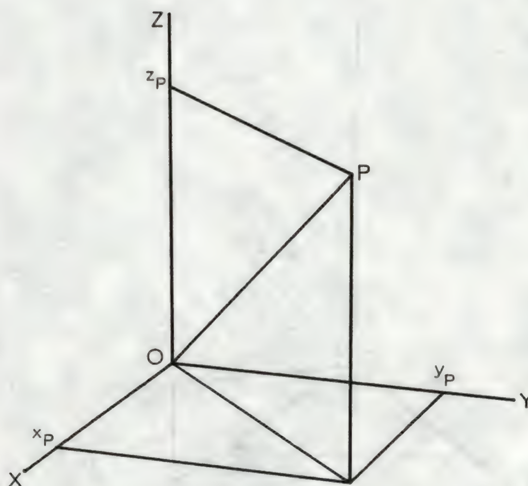
Dit boek is ook in het Engels verschenen onder de titel *Interactive 3D Computer Graphics* (John Wiley & Sons, Chichester).

L. Ammeraal
april 1988

1 HET ONTWERPEN IN DRIE DIMENSIES

1.1 DRIEDIMENSIONALE COÖRDINATEN

Dit boek gaat over D3D, wat betekent 'Designing in 3 Dimensions', ofwel 'het ontwerpen in drie dimensies'. De programmatekst van D3D, geschreven in Turbo C, is opgenomen en wordt besproken in dit boek. Gelukkig hoeft u het programma niet zelf over te typen, maar kunt u in plaats daarvan bij de uitgever een tweetal diskettes bestellen waarop zowel de programmatekst als de vertaalde versie (D3D.EXE) staat. Dank zij het laatste kunt u het programma gebruiken zonder bekend te zijn met de programmeertaal C en zonder zelfs van een C-compiler gebruik te maken. Als u precies wilt weten hoe de programma's in dit boek werken, dan zult u waarschijnlijk interessant materiaal vinden in de hoofdstukken 3 en 4, die bekendheid met C veronderstellen. Daarentegen is geen programmeerkennis vereist voor de hoofdstukken 1 en 2, die geschreven zijn voor D3D-*gebruikers*. Wel is enige elementaire wiskundekennis nodig, zoals uit de rest van deze paragraaf zal blijken.

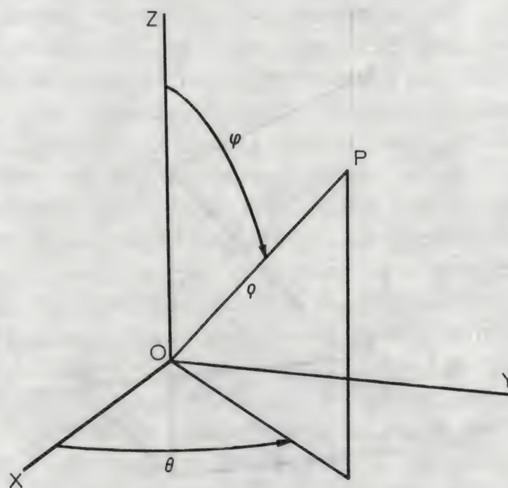


Figuur 1.1 Rechthoekige coördinaten

In de wiskunde en de techniek wordt veel gebruik gemaakt van een driedimensionaal coördinatenstelsel met een x-as, een y-as en een z-as, zoals figuur 1.1 laat zien. We noemen dit een *rechthoekig* (of Cartesisch) coördinatenstelsel: elke twee van deze drie assen staan loodrecht op elkaar. De drie assen gaan door de oorsprong O en zijn oneindig lang. In figuur 1.1 zijn alleen kleine stukjes van drie halve assen getekend; we noemen deze de *positieve* assen. Coördinaten zijn reële getallen. Punt P heeft de (rechthoekige) coördinaten x_P , y_P en z_P , wat betekent dat we, beginnend in O, punt P kunnen bereiken door eerst een afstand x_P langs de positieve x-as, dan een afstand y_P in de richting van de positieve y-as en, tenslotte, een afstand z_P in de positieve z-richting af te leggen.

We zeggen dat het coördinatenstelsel in figuur 1.1 een *rechts* coördinatenstelsel is. Dit betekent het volgende. Beschouw een rotatie van de positieve x-as over een hoek van 90° om de z-as, zodanig dat de x-as na deze rotatie samenvalt met de huidige y-as en vergelijk deze rotatie met het draaien met een rechtse (dat is een normale) schroef. De schroef zal zich dan langzaam een stukje voorwaarts bewegen in de richting van de positieve z-as. Er zijn uiteraard verschillende manieren waarop we een rechts coördinatenstelsel in de ruimte kunnen plaatsen. Zoals figuur 1.1 laat zien, laten we de positieve z-as omhoog wijzen, zodat de x- en de y-as in een horizontaal vlak, het zogenaamde *xy-vlak*, liggen.

Naast rechthoekige coördinaten zijn ook *bolcoördinaten* nuttig voor ons doel. Ook hierbij leggen we de positie van een punt vast met behulp van drie getallen, maar in plaats van x_P , y_P en z_P (of kortweg x, y en z) gebruiken we nu de Griekse letters ρ, θ, en φ (rho, theta en phi). Zoals Figuur 1.2 laat zien, is ρ de afstand tussen P en O, of,



Figuur 1.2 Bolcoördinaten

met andere woorden, het is de straal van de bol die O als middelpunt heeft en door P gaat.

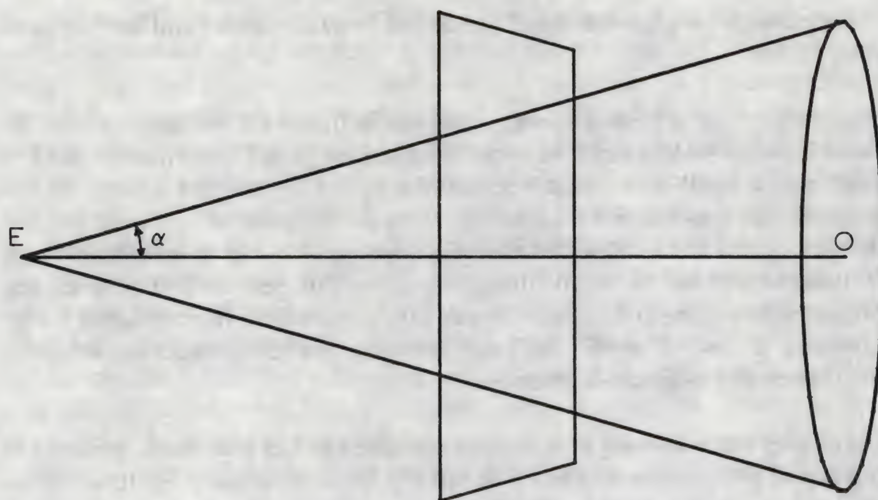
De symbolen θ en φ stellen hoeken voor, die in figuur 1.2 zijn aangegeven. We meten θ in het xy-vlak, waarbij we gebruik maken van punt P', de projectie van P in dit vlak: we vinden P' door vanuit P een loodlijn op het xy-vlak neer te laten. Bij een draaiing om de z-as is dan θ de hoek waarover de positieve x-as (in positieve zin) geroteerd zou moeten worden totdat hij gaat door punt P'. (Met 'positieve zin' wordt hier bedoeld dat de overeenkomstige draaiing van een rechtse schroef een voortgaande beweging in de richting van de positieve z-as zou veroorzaken.) Zo ligt θ bijvoorbeeld tussen 0° en 90° als P' zich bevindt in het eerste kwadrant, dat is het gebied tussen de positieve x-as en de positieve y-as.

De betekenis van φ kan vrij kort worden omschreven: het is de hoek, gemeten in een verticaal vlak, tussen de z-as en de lijn OP. De waarde van φ ligt tussen 0° en 180°. Als u met de trigonometrische functies *sinus* en *cosinus* bekend bent, dan zult u in staat zijn het volgende verband tussen de bolcoördinaten ρ, θ en φ enerzijds en de rechthoekige coördinaten x, y en z anderzijds te verifiëren:

$$x = \rho \sin \varphi \cos \theta$$
$$y = \rho \sin \varphi \sin \theta$$
$$z = \rho \cos \varphi.$$

1.2 VOORWERP, OOGPUNT EN PERSPECTIVISCH BEELD

Bij het gebruik van D3D zullen we ons bezighouden met punten, lijnstukken, en massieve voorwerpen (in de wiskunde vaak *lichamen* genoemd) die begrensd worden door platte vlakken. Gekromde oppervlakken kunnen we benaderen door middel van een verzameling platte grensvlakken, analoog aan de benadering van een kromme door middel van een aantal rechte lijnstukken. Zo'n grensvlak kan een willekeurige veelhoek zijn, waar mogelijk gaten in kunnen zitten. We zullen op een of andere manier de positie van de hoekpunten van deze veelhoeken moeten kunnen opgeven; voor dit doel gebruiken we rechthoekige coördinaten in een rechts coördinatenstelsel, zoals besproken in paragraaf 1.1. We hoeven ons er geen zorgen over te maken of het gewenste perspectivische beeld van een af te beelden voorwerp wel zal passen binnen de kaders van ons beeldscherm. Aangezien het zogenaamde schalen en positioneren volledig automatisch wordt gedaan, kunnen we een willekeurige eenheid van lengte gebruiken, zoals bijvoorbeeld een mm, een m of een inch. Wel moeten we hierbij consequent zijn en dezelfde eenheid gebruiken in alle richtingen. Dit betreft ook de manier waarop we de positie van ons oog (het oogpunt) opgeven. Laten we de letter E (afkomstig van het Engelse woord *eye)* gebruiken om dit punt aan te duiden. Punt E is van belang in relatie tot een enigszins centraal in het voorwerp gekozen punt O. (De letter O is de eerste letter van *object*,



Figuur 1.3 Kegel en kijkrichting EO

het Engelse woord voor *voorwerp*; bovendien zullen we punt O straks ook gebruiken als de oorsprong van een coördinatenstelsel). De richting EO is dan de kijkrichting. Zoals figuur 1.3 illustreert, kunnen we vanuit E alles zien binnen een zekere kegel, waarvan E de top en CE de as is.

Om het oogpunt E relatief ten opzichte van het voorwerp te kunnen opgeven stellen we ons een nieuw coördinatenstelsel voor, waarvan het centrale punt O de oorsprong is en waarvan de assen evenwijdig zijn met de overeenkomstige oorspronkelijke assen. We geven dan de bolcoördinaten ρ, θ en φ van oogpunt E op ten opzichte van dit nieuwe coördinatenstelsel. Daarbij is dus ρ de lengte van lijnstuk OE, zie ook figuur 1.3, met andere woorden, ρ stelt de *kijkafstand* voor. Figuur 1.3 toont ons ook het zogenaamde *tafereel*, een vlak loodrecht op de kijkrichting. Alle zichtbare punten van het voorwerp zenden lichtstralen naar het oogpunt E. De snijpunten van deze lichtstralen met het tafereel vormen het perspectivisch beeld waarin we geïnteresseerd zijn. Deze manier om een voorwerp op een vlak te projecteren wordt ook wel *centrale projectie* genoemd, omdat alle projectielijnen gaan door oogpunt E, het projectiecentrum.

Het zal duidelijk zijn dat de afstand tussen het tafereel en oogpunt E de grootte van het beeld bepaalt. Programma D3D zal zelf deze afstand zo kiezen dat het beeld van het hele voorwerp netjes past binnen de grenzen van het beeldscherm, zodat wij ons niet over de positie van het tafereel hoeven te bekommeren.

Hoek α tussen de as van de kegel en een rechte lijn op het kegeloppervlak moet niet al te groot zijn als we een redelijk perspectivisch beeld willen verkrijgen. We

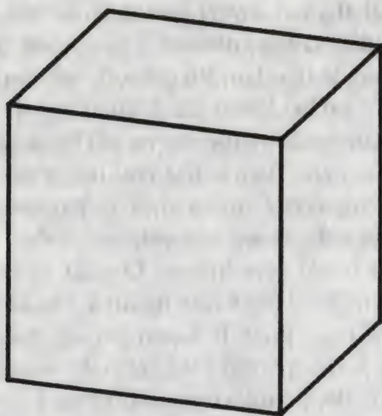
bereiken dit door de kijkafstand ρ veel groter te nemen dan de grootte van het voorwerp. Als we bijvoorbeeld een kubus met zijden van lengte 1 willen afbeelden, dan is een waarde $\rho \geq 5$ aan te bevelen. Bij gebruik van een heel grote waarde, zoals bijvoorbeeld $\rho = 1000000$, zal het beeld niet kleiner zijn dan wanneer we ρ tamelijk klein kiezen, dus het lijkt er misschien op dat we het beste altijd zo'n gigantisch grote waarde voor ρ kunnen nemen. Toch is dat niet het geval. Als ρ erg groot is, dan krijgen we geen echt perspectief, maar dan zullen evenwijdige lijnen van het voorwerp (zoals bijvoorbeeld een stel evenwijdige ribben van een kubus) ook als evenwijdige lijnen in het beeld verschijnen. Om dit te begrijpen moeten we ons realiseren dat in het laatste geval hoek α in figuur 1.3 zo klein is dat alle lichtstralen die vanuit het voorwerp naar punt E lopen vrijwel evenwijdig zijn. Zodoende benaderen we *parallelle projectie*, die veel gebruikt wordt in de praktijk van het technisch tekenen omdat deze projectiemethode veel gemakkelijker is dan echt perspectief. In figuur 1.4 zien we drie voorstellingen van een kubus met ribben van lengte 1, waarbij verschillende waarden voor de kijkafstand ρ zijn gebruikt.

De meeste lezers zullen de voorkeur geven aan figuur 1.4(a), waarbij $\rho = 5$ is gebruikt. We zien dat ver weg gelegen lijnstukken in de figuur verkort worden weergegeven, wat ons helpt bij het interpreteren van een ingewikkelde afbeelding (vooral als het voorwerp een draadmodel is). Figuur 1.4(b), waarbij $\rho = 100000$, is niet echt fout, maar eerder enigszins 'saai', omdat het perspectivisch effect ontbreekt. Evenwijdige ribben van de kubus verschijnen in deze figuur als evenwijdige lijnen en ver weg gelegen lijnstukken worden niet verkort weergegeven. (Er treedt weliswaar verkorting op, maar dit heeft alleen te maken met de richting van de desbetreffende lijnstukken, niet met hun afstand tot het oogpunt.) Strikt genomen is er nog wel enig perspectivisch effect overgebleven omdat ρ, hoewel heel groot, niet oneindig is, dus er is theoretisch een klein verschil in de lengte van de beelden van evenwijdige ribben. Maar zulke verschillen zijn te klein om te worden waargenomen, dus we zullen ze verwaarlozen en zeggen dat elke twee evenwijdige ribben in de beeldfiguur dezelfde lengte hebben en onderling evenwijdig zijn. We kunnen zodoende parallelle projectie als een bijzonder geval van perspectivische (ofwel centrale) projectie beschouwen, in plaats van als een fundamenteel ander onderwerp. Het betekent dat we met programma D3D niet alleen perspectivische beelden maar ook de meer conventionele afbeeldingen, opgeleverd door parallelle projectie, kunnen verkrijgen, zoals figuur 1.4(b) demonstreert.

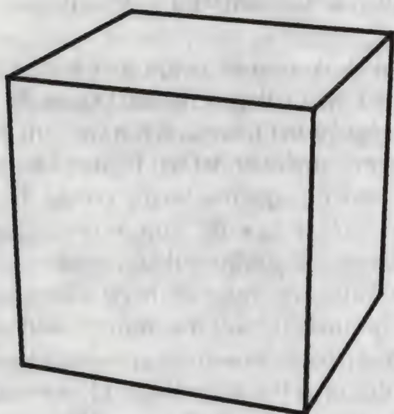
De kubus in figuur 1.4(c), verkregen met $\rho = 2$, lijkt onnatuurlijk. Hier hebben we te maken met een overdreven perspectivisch effect, omdat ρ te klein is. We zien hieraan dat we ρ ook niet te klein moeten kiezen.

Voor de hoeken θ en φ kan elke waarde worden gekozen. Als we ρ en φ constant nemen en θ laten toenemen van 0° tot 360°, dan is het alsof ons oog in een horizontaal vlak om het voorwerp heen draait. Als we $\varphi = 0°$ kiezen, dan ligt het oogpunt verticaal boven het voorwerp, terwijl bij $\varphi = 90°$ de kijkrichting horizontaal

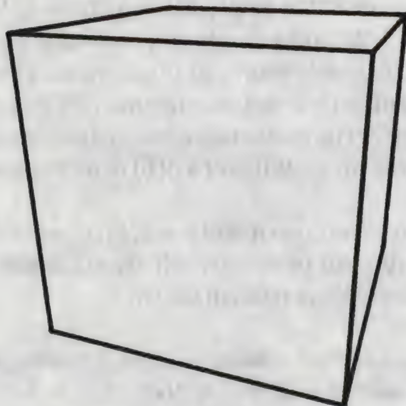
(a)



(b)



(c)



Figuur 1.4 (a) $\rho = 5$, *(b)* $\rho = 100\,000$, *(c)* $\rho = 2$

is. In de meeste praktische toepassingen gebruiken we een waarde van φ tussen 45° en 90°, zodat ons oog zich iets hoger dan het voorwerp bevindt. Dit is ook het geval in figuur 1.4, waar drie keer de hoeken $\theta = 20°$ en $\varphi = 70°$ zijn gebruikt.

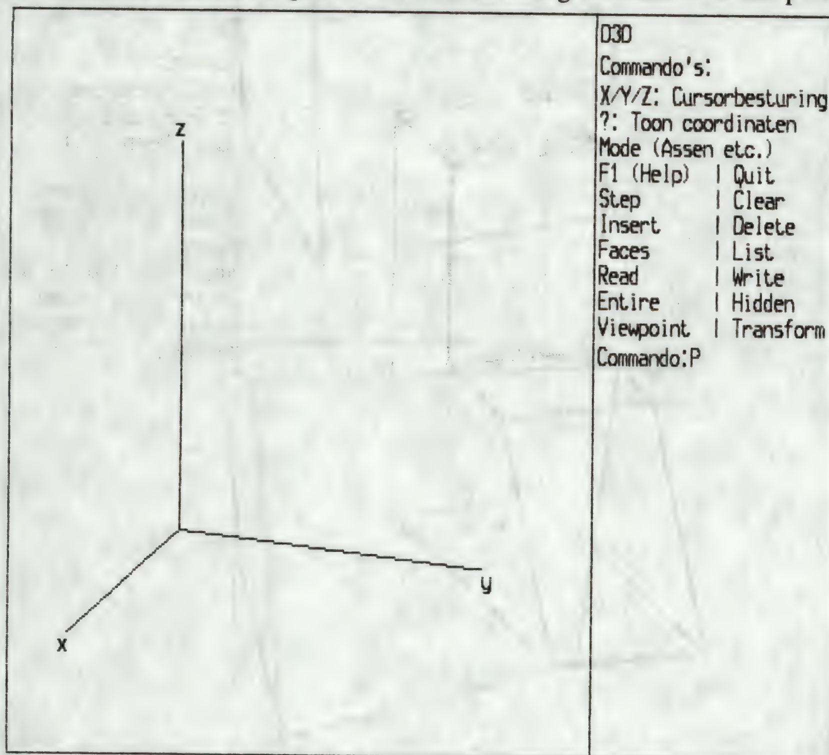
Als de diskettes die bij dit boek behoren (en via de uitgever verkregen kunnen worden) in uw bezit zijn, dan kunt u bijvoorbeeld de file EXAMPLE1.DAT gebruiken om met verschillende posities van het oogpunt te experimenteren. We starten het programma door het typen van

```
D3D
```

of, equivalent hiermee,

```
d3d
```

In de rest van het boek zullen we voor commando's in de regel hoofdletters gebruiken. Op deze wijze worden commando's duidelijk onderscheiden van gewone tekst, die hoofdzakelijk uit kleine letters bestaat. Steeds kunt u voor commando's desgewenst kleine letters in plaats van hoofdletters gebruiken als u dat prettiger



Figuur 1.5 Beginscherm

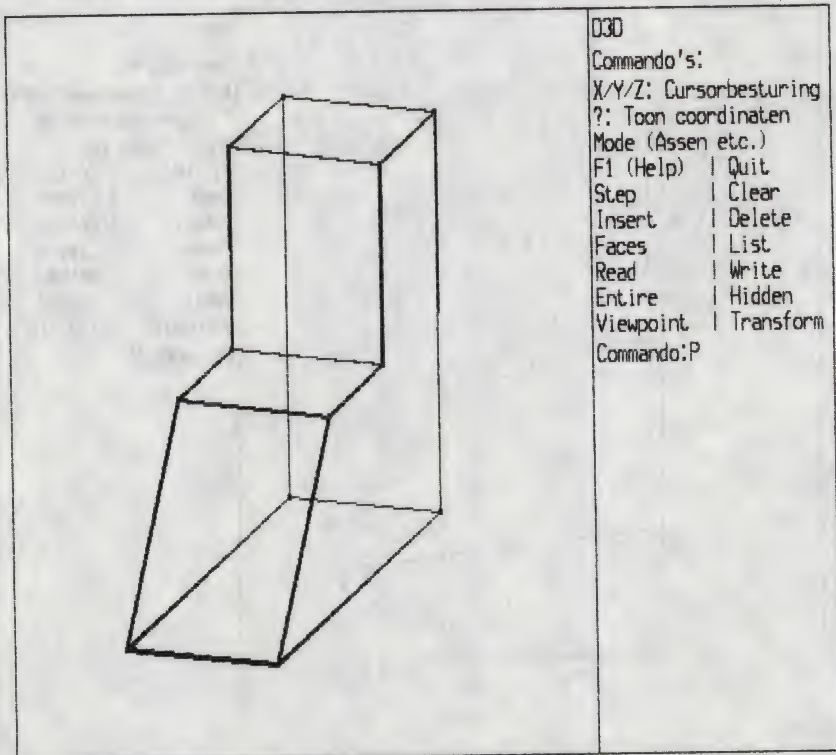
vindt. Na het starten van D3D verschijnt enige algemene informatie op het scherm. Als u deze gelezen heeft drukt u op de Enter-toets. Het scherm wordt dan zoals figuur 1.5 laat zien. Het is verdeeld in twee stukken: het linker gedeelte is bestemd voor het driedimensionale beeld waarin we geïnteresseerd zijn en rechts wordt een commandomenu zichtbaar; ook ingetypte commando's verschijnen op de rechter helft van het scherm.

Om snel tot een resultaat te komen typen we nu

R

dat wil zeggen *Read* (ofwel 'Lees'). Dit commando dient om alle gegevens betreffende een voorwerp te lezen van een file, vandaar dat de volgende boodschap op het scherm verschijnt:

```
Invoerfile:
```



Figuur 1.6 (a) Perspectief

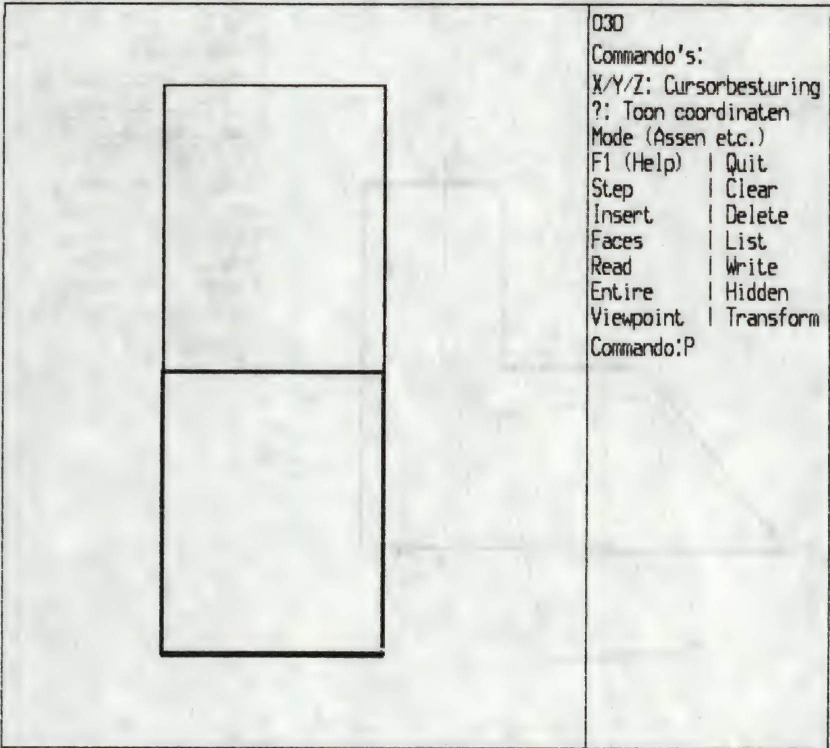
Dit betekent dat gevraagd wordt de naam van een file met invoergegevens in te typen. Als we nu typen

```
EXAMPLE1.DAT
```

(en daarna op de Enter-toets drukken) dan verkrijgen we figuur 1.6 (a) op het scherm.

Zoals u wellicht heeft opgemerkt, zijn lijnen van het voorwerp dicht bij het oogpunt dikker afgebeeld dan veraf gelegen lijnen. Op deze manier kunnen we draadmodellen gemakkelijker interpreteren dan wanneer alle lijnen in de figuur dezelfde dikte hebben. Het is waar dat het effect niet altijd even geslaagd is, maar gelukkig kunnen we, als we dat willen, ervoor zorgen dat alle lijnen in de figuur dezelfde dikte hebben, zoals we zullen zien in paragraaf 1.5.

Nadat we figuur 1.6(a) geproduceerd hebben kunnen we andere beelden van hetzelfde voorwerp verkrijgen door het oogpunt te verplaatsen. De positie van dit punt wordt opgegeven door middel van bolcoördinaten, zie figuur 1.2. Direct na het

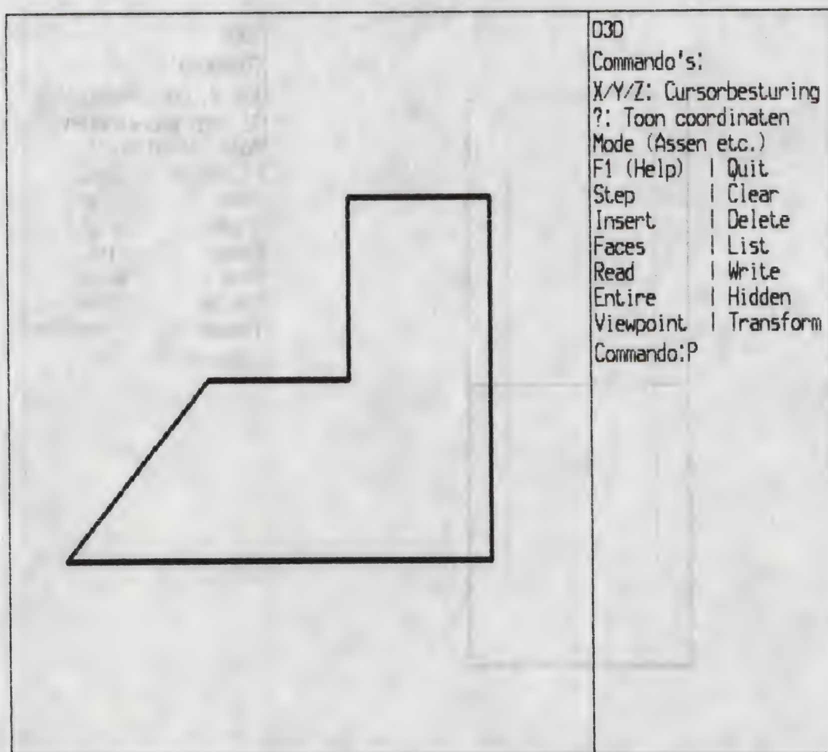


Figuur 1.6 (b) Vooraanzicht

starten van het programma heeft het oogpunt de ('default') coördinaten $\rho = 1000$, $\theta = 20°$, $\varphi = 75°$. We kunnen het oogpunt overal in de ruimte plaatsen (mits buiten het voorwerp) en aldus allerlei afbeeldingen produceren. Om van oogpunt te veranderen gebruiken we het commando

V

Het scherm verandert dan en gaat lijken op figuur 1.2, zodat we de betekenis van de bolcoördinaten ρ, θ en φ voor ons zien, voor het geval dat we die niet meer precies weten. Hun huidige waarden verschijnen achtereenvolgens in digitale vorm. Steeds moeten we nu een nieuwe waarde intypen of op de Enter-toets drukken, waarbij het laatste betekent dat we de getoonde huidige waarde accepteren. Let erop dat de hoeken θ en φ uitgedrukt moeten worden in graden. Zodra ρ, θ en φ op deze manier behandeld zijn, verschijnt het perspectivische beeld dat met het gegeven oogpunt correspondeert. Het is goed om eens een beetje met het programma te spelen en allerlei beelden van het voorwerp te produceren. We hebben gezien, dat we, als we erop staan, het effect van parallelle projectie kunnen verkrijgen, simpelweg door ρ heel groot te kiezen. Bovendien kunnen we gemakkelijk een voor-, een zij- en een



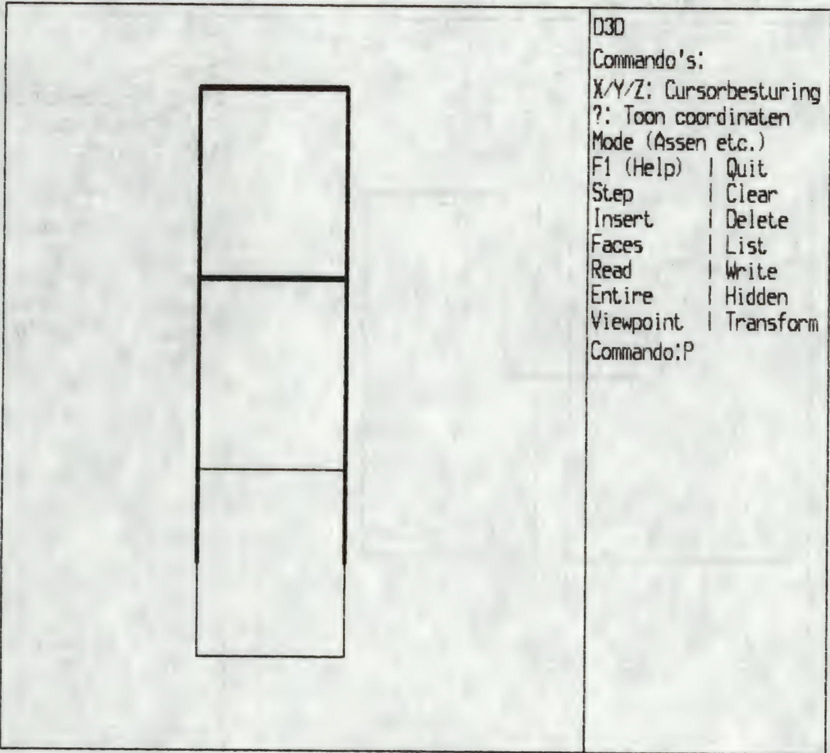
Figuur 1.6 (c) Zijaanzicht

bovenaanzicht produceren, zoals die ook vaak voorkomen in technische tekeningen. Deze drie aanzichten vindt u ook in de Figuren 1.6(b), (c) en (d). Het enige dat we ervoor hoeven te doen is een heel grote waarde voor ρ te nemen en θ en φ als volgt te kiezen:

Vooraanzicht: $\theta = 0°, \varphi = 90°$
Rechter zijaanzicht: $\theta = 90°, \varphi = 90°$
Bovenaanzicht: $\theta = 0°, \varphi = 0°$.

Deze aanzichten worden in de techniek veel gebruikt, niet alleen omdat zij gemakkelijk met de hand te tekenen zijn maar ook omdat alle lijnen loodrecht op de kijkrichting op ware lengte (niet verkort) verschijnen.

In deze paragraaf hebben we gezien dat we het programma kunnen 'binnenkomen' door de naam D3D in te typen. We moeten natuurlijk ook weten hoe we het moeten 'verlaten'. Zoals ook bij veel andere programma's het geval is, gebruiken we hiervoor het commando *Quit*, afgekort tot *Q*.



Figuur 1.6 (d) Bovenzicht

1.3 PUNTMANIPULATIE EN CURSORBESTURING

In plaats van een bestaand voorwerp van een file te lezen, zoals we dat in de vorige paragraaf hebben gedaan, zullen we nu zelf een voorwerp gaan ontwerpen en tekenen. Steeds zullen we rechte lijnstukken gebruiken om driedimensionale voorwerpen te tekenen. Dit is alleen mogelijk als de beide eindpunten van elk lijnstuk bekend zijn, dus we moeten het eerst hebben over *punten*. Voor het definiëren van een punt zullen we commando *Insert* gebruiken. Zodra we hiervan de eerste letter

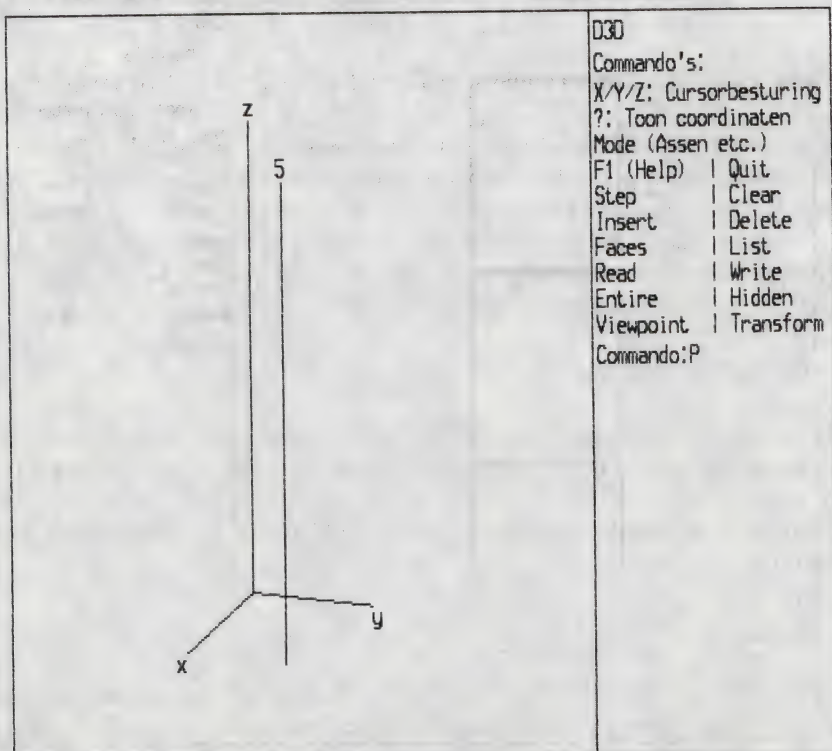
```
I
```

hebben ingetypt, wordt als volgt om vier getallen gevraagd:

```
n  x  y  z
```

We kunnen nu bijvoorbeeld intypen:

```
5  1.49  0.8  3
```



Figuur 1.7 Punt 5, toegevoegd met commando I

Nadat we de 3 aan het eind van deze regel hebben getypt moet de computer weten dat dit het laatste cijfer van de regel is, zodat we op de Enter-toets moeten drukken. Van nu af zullen we dat in situaties als deze niet expliciet vermelden. Steeds als u erg lang moet wachten en er niets lijkt te gebeuren, kunt u proberen of het drukken op de Entertoets helpt. De zojuist ingetypte getallen x, y en z zijn de rechthoekige coördinaten van het nieuwe punt. Getal n is gewoon een positief geheel getal waarmee we het punt later kunnen identificeren. We kunnen tenslotte een punt veel gemakkelijker met een enkel nummer n dan met drie coördinaten x, y en z aanduiden. Het toegevoegde punt wordt nu zichtbaar gemaakt, zoals figuur 1.7 laat zien. Behalve dit punt zien we ook een loodlijn die uit het punt op het xy-vlak wordt neergelaten. Om duidelijk te kunnen zien waar punt (x, y, z) zich in de driedimensionale ruimte bevindt is het punt verbonden met punt $(x, y, 0)$. Ook het puntnummer 5 is zichtbaar in de figuur.

Tijdens het construeren van een nieuw voorwerp willen we gewoonlijk dat puntnummers en assen zichtbaar zijn, terwijl we ze niet willen zien in het eindresultaat. We kunnen ze altijd zichtbaar of onzichtbaar maken door middel van commando

```
M
```

zoals we zullen zien aan het eind van paragraaf 1.4.

Als we veel punten moeten toevoegen, dan hoeven we commando *I* niet voor elk punt afzonderlijk te gebruiken. Programma D3D onthoudt dat we punten aan het toevoegen zijn zolang we geen ander commando gebruiken. Wanneer we wel een ander commando hebben gebruikt dan kunnen we weer verder gaan met het toevoegen van punten door opnieuw commando *I* te typen; als we dat vergeten dan zal het eerste ingetypte cijfer de foutboodschap

```
Begin niet met cijfer
```

tot gevolg hebben. (Dit moge intolerant lijken, maar het is de beste manier om verwarring met commando *F* te vermijden, zoals we zullen zien in paragraaf 1.4.)

Punten kunnen worden verwijderd door middel van het commando *Delete*, afgekort

```
D
```

Na deze letter getypt te hebben, kunnen we alle punten verwijderen waarvan de nummers tussen twee grenzen liggen: we dienen een onder- en een bovengrens op te geven. Als slechts een enkel punt moet worden verwijderd dan typen we het nummer ervan in antwoord op:

```
Ondergrens:
```

en drukken eenvoudig op de Enter-toets wanneer hierna de regel

```
Bovengrens:
```

verschijnt. Als het opgegeven bereik van puntnummers tenminste één gedefinieerd punt bevat, dan verschijnt een nieuw beeld op het scherm, waarin de verwijderde punten niet meer voorkomen.

Het zou heel vervelend zijn als een compleet voorwerp zou worden vernietigd doordat we per ongeluk de toets *D* indrukken en niet weten wat we verder moeten doen. Het commando *D* is daarom erg veilig gemaakt: als voorgestelde ondergrens verschijnt een getal op het scherm dat 1 hoger is dan het hoogste puntnummer dat in gebruik is, zodat de lege verzameling van punten (dat wil zeggen geen enkel punt) wordt verwijderd als we na *D* twee keer op de Enter-toets drukken. (In ingewikkelde situaties, waarin niet alle puntnummers leesbaar op het scherm kunnen verschijnen, kunnen we commando *D* op een oneigenlijke manier gebruiken om aan de weet te komen wat het hoogste van alle gebruikte puntnummers is: na *D* lezen we de voorgestelde ondergrens af en verlagen dit met 1; daarna drukken we twee keer op de Enter-toets.)

Als we een heleboel punten hebben ingevoerd dan kan het zijn dat we de coördinaten van een zeker punt willen weten. We typen dan het vraagteken

```
?
```

hetgeen de volgende vraag doet verschijnen:

```
Puntnummer?
```

Nadat we het nummer van het punt in kwestie hebben ingetypt verschijnen de gevraagde coördinaten (de waarden van x, y en z) op het scherm.

Er is een andere manier om punten in te voeren, die in de meeste gevallen aantrekkelijker is dan het expliciet opgeven van coördinaten. We kunnen gebruik maken van een *cursor*, die op het scherm te zien is als een klein vierkantje. In tweedimensionale grafische toepassingen wordt wel gebruik gemaakt van de vier pijltjestoetsen (links, rechts, omhoog, omlaag) om de cursor in de vier overeenkomstige richtingen te bewegen. Nu we driedimensionaal werken willen we de cursor eigenlijk in zes in plaats van vier richtingen laten bewegen, want we hebben drie assen en we moeten in staat zijn de cursor op elke as zowel in de positieve als in de negatieve richting te bewegen. In D3D is dit probleem opgelost met behulp van de toetsen *X*, *Y*, *Z*, in combinatie met + en −. Op de meeste

toetsenborden vinden we twee plus- en mintekens. Het is het handigst die te gebruiken welke zich dichtbij de pijltjestoetsen bevinden, omdat we dan voor het plusteken de Shift-toets niet hoeven in te drukken (wat wel het geval is met het andere plusteken, dat een toets deelt met het gelijkteken). Zodra we een van de toetsen *X*, *Y* en *Z* indrukken, verschijnt de cursor in de oorsprong van het coördinatenstelsel. Zolang we de cursor willen bewegen langs de as die met de gekozen toets overeenkomt kunnen we de toetsen + en − gebruiken. Om van richting te veranderen (anders dan van + naar − en omgekeerd), typen we weer de naam van de gekozen asrichting, enzovoort.

De standaard ('default') stapgrootte voor de cursor is 0.2; indien gewenst, kunnen we deze veranderen door *S* te typen. Veronderstel bijvoorbeeld dat we het punt (0.6, 0.4, -0.2) willen definiëren. Dit kan door achtereenvolgens de volgende toetsen in te drukken:

```
X + + + Y + + Z - I
```

Het volgende tabelletje toont de opeenvolgende cursorposities:

Ingedrukte toets	Cursor-posities		
	x	y	z
X	0.0	0.0	0.0
+	0.2	0.0	0.0
+	0.4	0.0	0.0
+	0.6	0.0	0.0
Y	0.6	0.0	0.0
+	0.6	0.2	0.0
+	0.6	0.4	0.0
Z	0.6	0.4	0.0
−	0.6	0.4	−0.2
I	0.6	0.4	−0.2

Door *I* te typen wordt de dan geldende cursorpositie gebruikt voor de definitie van een nieuw punt. Dit punt krijgt automatisch een puntnummer, dat op het beeldscherm wordt getoond. Er wordt eenvoudig het laagste positieve gehele getal genomen dat nog niet als puntnummer in gebruik is. Door *I* te typen verandert de cursorpositie niet en de geldende richting (*Z* in ons voorbeeld) evenmin. Als dus bijvoorbeeld (0.6, 0.4, −0.4) de gewenste volgende cursorpositie is, dan hoeven we alleen maar één keer op de mintoets te drukken.

Bovenstaand voorbeeld lijkt ingewikkelder dan het in werkelijkheid is. Ten eerste wordt steeds de huidige cursorpositie in digitale vorm op het scherm weergegeven,

dus in plaats van nauwlettend de cursorstappen te tellen kunnen we eenvoudig de coördinaten van de cursor aflezen en veranderen zoals we dat wensen. Belangrijker is dat we in veel toepassingen niet beginnen met gegeven coördinaten om daarmee een punt in de ruimte te definiëren, maar dat het net andersom gaat. We willen ergens in de driedimensionale ruimte een aantal punten (en andere meetkundige objecten) plaatsen, zodanig dat het geheel er goed uitziet; daarna gebruiken we de coördinaten van de gekozen punten om de positie ervan vast te leggen zodat we de situatie kunnen analyseren en bespreken.

Het kan soms wenselijk zijn de cursor te plaatsen in een punt dat we tevoren ingevoerd hebben. De snelste manier om dit te doen is de cursor naar het punt toe te bewegen totdat hij (vergeleken met alle overige gedefinieerde punten) er vlak bij is en dan

```
J
```

te typen. Dit veroorzaakt een sprong ('jump') van de cursor naar het dichtstbijgelegen gedefinieerde punt, dat dan het gewenste punt is. Dit bespaart ons de moeite van het handmatig bewegen van de cursor tot hij precies in het gewenste punt is.

We hebben gezien dat er in feite twee commando's *I* zijn, namelijk een op het niveau van het hoofdmenu de ander in 'cursorbesturingsmode' (dat wil zeggen wanneer de cursor op het scherm te zien is). Dit is ook van toepassing op het commando

```
D
```

dat gebruikt wordt om een punt te verwijderen. We kunnen evenwel commando *D* in cursorbesturingsmode alleen vlak na commando *J* gebruiken, hetgeen garandeert dat de cursor zich exact bevindt in het punt dat we willen verwijderen. Verder kunnen we nu slechts één punt tegelijk verwijderen.

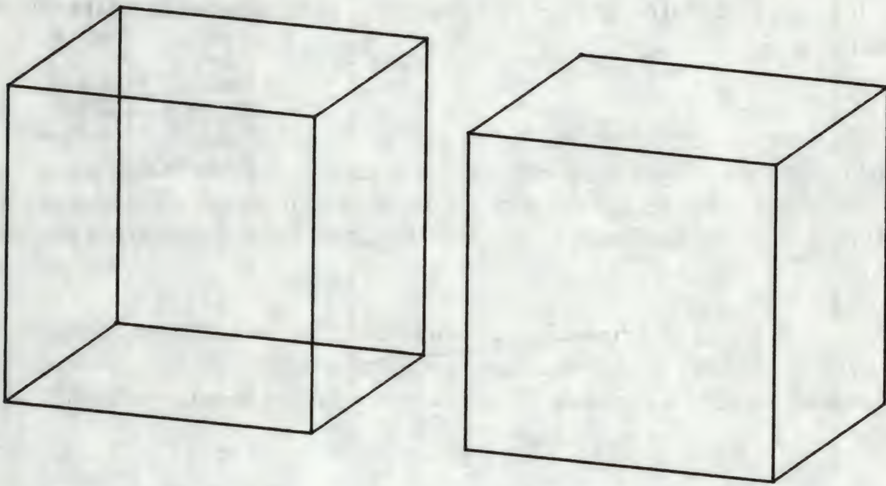
Als we alle gedefinieerde punten willen verwijderen, dan hoeven we dat niet één voor één te doen, maar dan kunnen we het hele scherm schoonmaken door middel van het commando

```
C
```

Dit commando is nuttig als we bijvoorbeeld voor de aardigheid zo maar wat punten hebben gedefinieerd om te zien hoe D3D werkt en we alle gedefinieerde punten willen verwijderen om met een schoon scherm te beginnen voor wat serieuzer werk. We hebben het ook nodig als we met commando *R* een voorwerp van een file hebben gelezen en we, na er een tijdje naar gekeken te hebben, zelf iets willen creëren, wat tenslotte meer voldoening geeft.

1.4 LIJNSTUKKEN, GRENSVLAKKEN EN VOORWERPEN

Driedimensionale voorwerpen kunnen op twee manieren worden afgebeeld, namelijk als draadmodellen en als (ondoorzichtige) massieve lichamen. Zoals figuur 1.8 laat zien betekent dit dat we van een kubus, gezien vanuit een willekeurig oogpunt, hetzij alle zes, hetzij slechts drie grensvlakken zien. Wat de ribben betreft, òf we zien ze alle twaalf òf er zijn er maar negen zichtbaar. We zullen beide soorten afbeeldingen gaan maken; hoewel we beginnen met draadmodellen, zullen we de voorwerpen op een zodanige manier opbouwen dat D3D in staat is onzichtbare lijnstukken te verwijderen wanneer we dat verlangen.

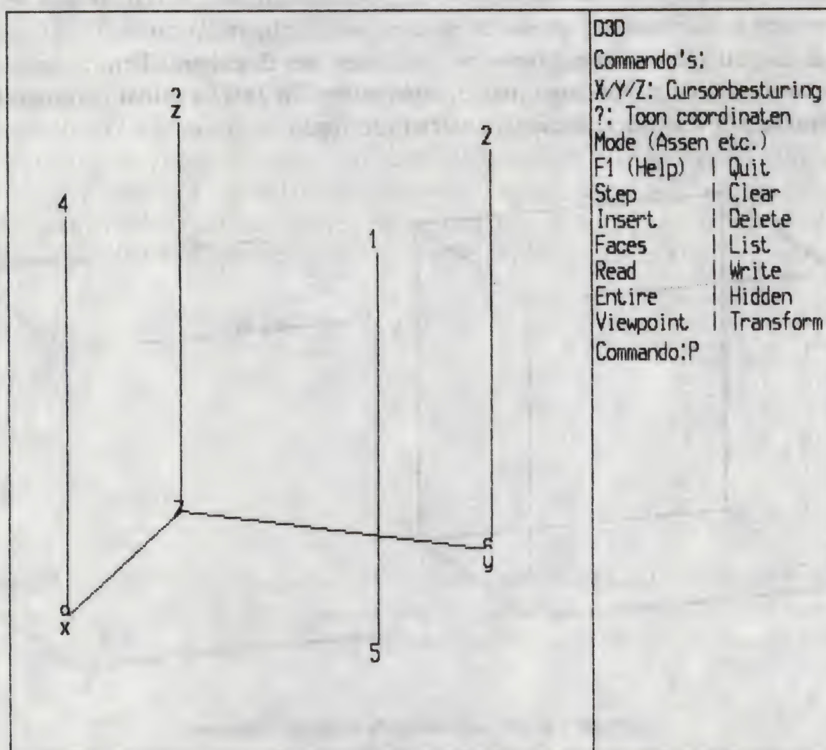


Figuur 1.8 Draadmodel en massief lichaam

Laten we aannemen dat we de acht hoekpunten van een kubus hebben gedefinieerd door ze in te voeren op een van de twee manieren besproken in paragraaf 1.3, dat wil zeggen hetzij door puntnummers en coördinaten expliciet op te geven, hetzij met behulp van cursorbesturing. In het eerstgenoemde geval zou het dus kunnen zijn dat we, na commando *I*, de volgende regels hebben ingetypt:

```
1 1 1 1
2 Ø 1 1
3 Ø Ø 1
4 1 Ø 1
5 1 1 Ø
6 Ø 1 Ø
7 Ø Ø Ø
8 1 Ø Ø
```

In de praktijk vormt cursorbesturing een prettiger middel om hetzelfde te bereiken. In beide gevallen zullen de gedefinieerde punten op het scherm zichtbaar zijn zoals aangegeven in figuur 1.9. (Uiteraard hangt hetgeen op het scherm verschijnt ook van de positie van het oogpunt af. In figuur 1.9 zijn de standaardwaarden $\rho = 1000$, $\theta = 20°$, $\varphi = 75°$ gebruikt.)



Figuur 1.9 Hoekpunten van een kubus

We definiëren lijnstukken en grensvlakken door het commando

```
F
```

(wat staat voor 'faces') in te typen, waarbij de volgende regel verschijnt:

```
Nrs., daarna een punt:
```

Als we alleen in draadmodellen geïnteresseerd waren, dan zouden we nu eerst de horizontale grensvlakken (door middel van hun vier hoekpunten) en dan de vier verticale ribben (door middel van hun twee eindpunten) kunnen opgeven en wel als volgt:

```
1 2 3 4.
5 6 7 8.
1 5.
2 6.
3 7.
4 8.
```

Deze eenvoudige invoergegevens zijn voldoende om draadmodellen van de kubus te tekenen. Er is evenwel een iets andere werkwijze vereist als we willen dat de invoergegevens op een zodanige manier worden opgeborgen dat D3D de kubus kan afbeelden als een massief lichaam. We moeten dan alle zes de grensvlakken van de kubus opgeven. Bovendien moet voor elk grensvlak de volgorde van de vier hoekpunten 'tegen de klok in' zijn, aannemende dat we de kubus van de buitenkant bekijken. Daarom geven we, in plaats van de bovenstaande zes regels, de voorkeur aan:

```
1 2 3 4.
8 7 6 5.
8 5 1 4.
5 6 2 1.
6 7 3 2.
7 8 4 3.
```

Als we bijvoorbeeld het ondervlak niet van de buitenkant (met ons oog onder de kubus) maar vanuit een meer comfortabele positie bekijken, zodat we het zien als in de figuren 1.8 en 1.9, dan correspondeert de rij

```
8 7 6 5
```

met de klokwijzerrichting (terwijl de tegenovergestelde richting is vereist). We kunnen zeggen dat we twee fouten hebben gemaakt. Ten eerste hebben we het ondervlak bekeken dwars door het (transparante) materiaal van de kubus in plaats van vanaf de buitenkant en ten tweede hebben we hoekpunten kloksgewijs in plaats van anti-kloksgewijs doorlopen. We kunnen er nu gebruik van maken dat deze twee 'fouten' elkaar opheffen. Dus steeds wanneer we het moeilijk vinden ons voor te stellen dat we een grensvlak van de buitenkant bekijken om te bepalen wat 'anti-kloksgewijs' betekent, dan kunnen we in plaats hiervan dat grensvlak van de 'verkeerde' kant bekijken, mits we tegelijkertijd de 'verkeerde', dat is de kloksgewijze, omloopszin kiezen. U zult zich misschien afvragen waarom de volgorde van de punten anti-kloksgewijs moet zijn. Er wordt uitsluitend gebruik van gemaakt in de algoritme voor het weglaten van onzichtbare lijnen (waarvoor we commando *H* gebruiken, zoals in de volgende paragraaf duidelijk zal worden). Zoals we net hebben gezien, hangt de omloopszin af van de manier waarop we een grensvlak bekijken: bekeken vanaf de verkeerde zijde verandert 'anti-kloksgewijs'

in 'kloksgewijs' en omgekeerd. Dit feit wordt door het programma gebruikt om snel te bepalen of een gegeven grensvlak, bekeken vanaf het gegeven oogpunt een zogenaamd *achtervlak* is. Als dat zo is dan wordt het eenvoudigweg genegeerd in het nogal tijdrovende proces van verborgenlijn-eliminatie, wat dit proces aanzienlijk versnelt. In ons voorbeeld van de kubus, met een oogpunt dat de figuren 1.8 en 1.9 oplevert, zijn er drie achtervlakken. (D3D gebruikt alleen de eerste drie hoekpunten van een grensvlak om te bepalen of het een achtervlak is. Dit impliceert dat het tweede opgegeven hoekpunt *convex* moet zijn, zoals we zullen zien in paragraaf 1.7.)

Er is een geval waarin de volgorde van de puntnummers volkomen onbelangrijk is, namelijk als het er maar twee zijn, zoals bijvoorbeeld in

```
F
2 8.
```

In ons voorbeeld komt dit overeen met een lichaamsdiagonaal (die twee tegenover elkaar gelegen hoekpunten van de kubus met elkaar verbindt). Een invoerregel met de nummers van slechts twee punten geeft dus het *lijnstuk* tussen deze twee punten aan. Als er precies drie puntnummers zijn dan definiëren die een driehoek, althans wanneer de gegeven punten niet op dezelfde lijn liggen. Met vier of meer punten kunnen zich allerlei problemen voordoen als de punten niet werkelijk de hoekpunten van een veelhoek zijn. Het zal duidelijk zijn dat de punten in hetzelfde vlak moeten liggen. Als dat niet zo is, dan verschijnt de boodschap:

```
Niet in hetzelfde vlak
```

Analoog aan commando *I*, besproken in de vorige paragraaf, hoeft commando *F* niet steeds herhaald te worden: zolang als we grensvlakken aan het invoeren zijn onthoudt D3D dat we commando *F* hebben gegeven. Na een ander commando te hebben gegeven moeten we opnieuw *F* intypen als we verder willen gaan met het invoeren van grensvlakken. Op deze wijze is de betekenis van een invoerregel beginnend met een getal altijd duidelijk: hij behoort òf bij commando *I* òf bij commando *F*. Als we niet met een van deze commando's bezig zijn dan zal een ingetypt cijfer de foutboodschap

```
Begin niet met cijfer
```

tot gevolg hebben. Denk eraan dat na het laatste hoekpunt van elk grensvlak een punt (.) moet worden getypt. De reden hiervoor is dat een grensvlak een willekeurige veelhoek kan zijn (in plaats van een vierkant in ons voorbeeld) en dat zo'n veelhoek zoveel hoekpunten kan hebben dat hun nummers niet op een enkele regel passen in de ruimte die daarvoor op het scherm beschikbaar is. In dat geval kunnen we de hoekpuntnummers over verschillende regels verdelen. Er moet een

signaal zijn dat het einde van de getallenrij aangeeft en aangezien we voor dit doel de Entertoets niet kunnen gebruiken is hiervoor de punt gekozen.

Wanneer we grensvlakken aan het invoeren zijn, kunnen we een fout maken, hetgeen gewoonlijk resulteert in een grensvlak dat we niet bedoeld hebben. We willen dan het zojuist ingevoerde vlak verwijderen. Het zou bovendien mooi zijn als we een grensvlak konden verwijderen op een soortgelijke manier als we met commando *D* een punt verwijderen. We kunnen ons afvragen wat er gebeurt als we een punt verwijderen dat gebruikt is om een vlak te definiëren. Als dat betekent dat zo'n vlak dan ook wordt verwijderd, dan kunnen we langs deze weg het gestelde doel (de verwijdering van een vlak) bereiken. Dit is inderdaad mogelijk, maar we moeten ons ervan bewust zijn dat het verwijderen van een punt impliceert dat *alle* grensvlakken waarvan dat punt een hoekpunt is, worden verwijderd, zodat in de meeste gevallen deze methode te rigoreus is: er kunnen grensvlakken door verwijderd worden die we willen behouden. Er is daarom een ander middel om grensvlakken te verwijderen. We kunnen typen

```
L
```

wat *List* betekent. Alle gedefinieerde vlakken verschijnen dan één voor één in de vorm van een rijtje puntnummers. Steeds wordt gevraagd of dit in orde is, zoals bijvoorbeeld in:

```
1 2 3 4.
OK? (J/N):
```

Typen we nu *N* in, dan wordt het vlak met de hoekpunten 1 2 3 4 verwijderd; door *J* te typen verandert er niets en in beide gevallen verschijnt een rijtje puntnummers voor het volgende vlak, enzovoort. Als dit proces willen laten stoppen dan drukken we op een andere toets dan *J* of *N*.

Nadat we alle gegevens van een voorwerp, zoals een kubus in ons voorbeeld, hebben ingevoerd, zal dat voorwerp op het scherm zichtbaar zijn. We kunnen nu de puntnummers en de assen (evenals de verticale projectielijnen) van het scherm verwijderen door van het 'mode'commando

```
M
```

gebruik te maken en *N* te typen in antwoord op de vragen

```
Puntnummers? (J/N):
Assen? (J/N):
```

(In werkelijkheid wordt elk antwoord dat verschilt van *J* opgevat als *N*.) Als we *J*

antwoorden op laatstgenoemde vraag, dan verschijnt de volgende vraag:

```
Hulplijnen? (J/N):
```

Deze vraag betreft de verticale projectielijnen die vanuit elk gedefinieerd punt worden neergelaten op het xy-vlak. Zij zijn nuttig om te zien waar 'losse' punten zich in de driedimensionale ruimte bevinden, maar meestal ongewenst als de punten onderling zijn verbonden door lijnstukken.

Er verschijnen nu de volgende twee vragen:

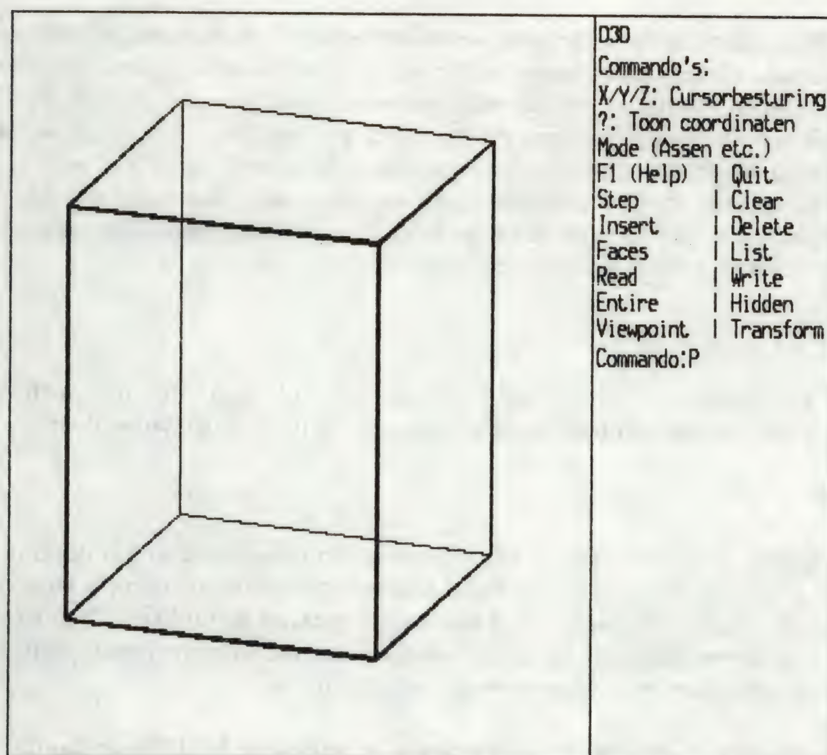
```
Dik en dun? (J/N):
Geheel scherm? (J/N):
```

Als onze antwoorden respectievelijk *J* en *N* zijn dan wordt het voorwerp opnieuw getekend, zoals afgebeeld in figuur 1.10. Er zijn nu geen assen in de afbeelding opgenomen, evenmin als assen of hulplijnen. Zoals vermeld in paragraaf 1.2 worden dichtbij gelegen lijnstukken dikker weergegeven dan lijnstukken die ver weg zijn. Het is gemakkelijker draadmodellen te interpreteren met deze wijze van weergeven dan wanneer alle lijnen even dun zijn. In sommige gevallen is het effect van verschillende lijndikte niet zo goed. Als we liever alle lijnen even dun willen hebben dan moeten we *N* intypen als antwoord op de eerste van de twee bovenstaande vragen. We kunnen het hele scherm (in plaats van alleen het linker deel) gebruiken om het voorwerp af te beelden door de tweede van bovenstaande twee vragen met *J* te beantwoorden. Dit is analoog aan het gebruik van commando *E*, dat in de volgende paragraaf wordt behandeld.

We kunnen nog vermelden dat de 'standaard mode', dat is de situatie onmiddellijk nadat we het programma gestart hebben door

```
D3D
```

te typen, overeenkomt met de antwoorden *J*, *J*, *J*, *J*, *N*, in die volgorde, op de bovengenoemde vijf vragen die na het commando *M* worden gesteld. Zodra we een compleet voorwerp op het scherm hebben verkregen, gebruiken we gewoonlijk commando *M* om puntnummers, assen en hulplijnen te verwijderen. De automatisch gekozen mode verschilt ook van de zojuist genoemde als we een compleet voorwerp van een file inlezen door middel van commando *R*, zoals we hebben besproken in paragraaf 1.2. Tenzij slechts een verzameling punten (zonder vlakken of lijnstukken) van een file wordt gelezen, ontbreken na gebruik van commando *R* puntnummers en assen, dus we zouden kunnen zeggen dat dit commando de 'standaard' mode verandert. Uiteraard kunnen we steeds zelf met commando *M* de mode veranderen en dus ook na *R* puntnummers enzovoorts zichtbaar maken.



Figuur 1.10 Kubus

1.5 VOLLEDIG SCHERM; VERBORGEN LIJNEN; AFDRUKKEN

We willen onze grafische resultaten nu afdrukken op een matrixprinter. Voordat we dit doen zullen we ervoor zorgen dat het hele scherm gebruikt wordt, zodat een groter beeld ontstaat. We bereiken dit door

```
E
```

(met de betekenis 'Entire screen') in te typen. Zoals we in de vorige paragraaf hebben gezien, kunnen we ook naar het volledige scherm overschakelen door middel van het *Mode*-commando (*M*), wat handig is als we tevens verandering willen brengen in het al of niet vertonen van puntnummers, assen, hulplijnen en variërende lijndikte. Als geen enkele van deze laatste aspecten (collectief 'mode' genoemd) verandert, dan is *E* handiger dan *M*, omdat eerstgenoemd commando ons de moeite van het herhaaldelijk *J* of *N* intypen bespaart. Voor zover het de omschakeling naar het volledige scherm betreft hebben de commando's *M* en *E*

hetzelfde effect. In beide gevallen verdwijnt het grafische scherm en krijgen we gelegenheid de verhouding tussen horizontale en verticale afmetingen op de printer af te stemmen. Als het scherm bijvoorbeeld een vierkant vertoont, dan willen we dit ook als een vierkant afdrukken; evenzo willen we voorkomen dat cirkels worden misvormd tot ellipsen. Omdat D3D eenvoudig zogenaamde 'pixels' van het scherm afbeeldt op 'dots' van de matrixprinter, kunnen we de verlichte pixels zo kiezen dat de uitvoer op de printer goed is. Dit gebeurt als we na commando *E* (wanneer ons die mogelijkheid wordt geboden) de letter

```
A
```

(met de betekenis: *Aspect ratio*) intypen. U zult dan misschien een wat teleurstellend resultaat op het beeldscherm krijgen, maar als u daarna, door

```
P
```

in te typen, de uitvoer naar de printer stuurt, dan verschijnt daar het beeld in de goede afmetingen. Als u een optimaal resultaat op het beeldscherm belangrijker vindt, dan dient u in plaats van *A* een andere toets in de drukken. U moet dus kiezen: of de verhoudingen zijn goed op het scherm maar incorrect op de printer, of zij zijn goed op de printer maar incorrect op het scherm.

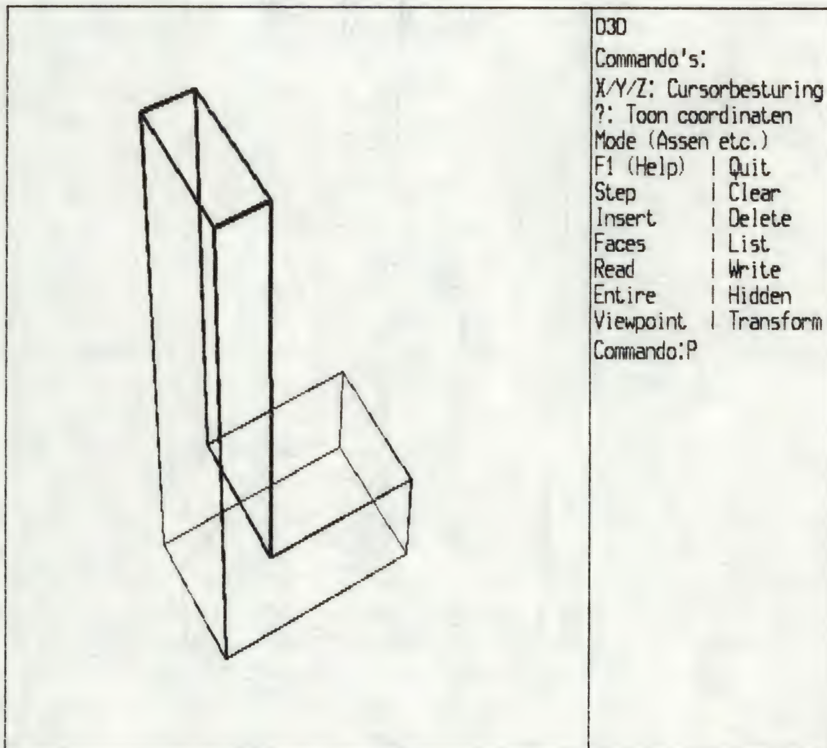
We kunnen in plaats van *A* ook *O* typen, waardoor de beeldfiguur die op het volledige scherm verschijnt tevens in een 'Output file' wordt geschreven. Deze file is in de eerste plaats bedoeld als invoer voor een ander programma, PLOTHP, waarmee we een tekening op een Hewlett-Packard plotter kunnen verkrijgen; deze mogelijkheid, die we in paragraaf 4.6 uitvoeriger zullen bespreken, is bijzonder interessant als zo'n plotter of een ander apparaat voor grafische uitvoer beschikbaar is. U zult hebben opgemerkt dat er in dit boek twee soorten illustraties zijn. Die waarin nogal rafelige schuine lijnen voorkomen zijn vervaardigd op een matrixprinter, terwijl die met veel strakkere lijnen geproduceerd zijn op een HP-plotter, op de manier die zojuist is genoemd. Het belangrijkste wat u in dit stadium dient te onthouden is dat de invoerfiles voor D3D enerzijds en de met commando *O* geproduceerde files anderzijds verschillend van structuur zijn (hoewel het beide gewone ASCII files zijn). Om verwarring te voorkomen zullen we de eerstgenoemde files *objectfiles* noemen en ze namen geven die eindigen op *.DAT*, terwijl we de laatstgenoemde files *plotfiles* noemen; de naam van een plotfile eindigt altijd op *.PLT.*

Nadat we, door *P* te typen, de afbeelding hebben afgedrukt, of nadat we op de Enter-toets hebben gedrukt als we geen uitvoer op de printer wensen, verdwijnt het vergrote beeld; het scherm wordt dan weer zoals het was voordat we commando *E* gaven.

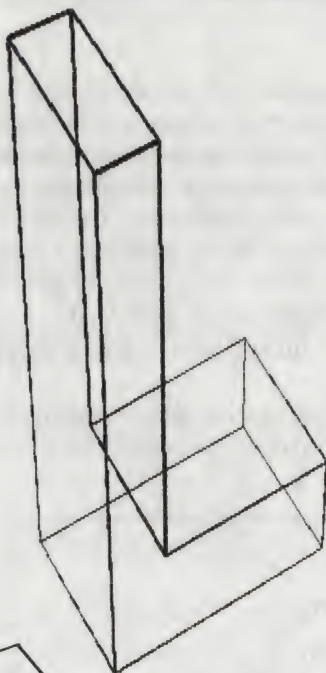
In principe heeft commando *E* tot gevolg dat het beeld dat al zichtbaar is op de linker helft van het scherm wordt vergroot, maar verder ongewijzigd blijft. Zoals we hebben gezien, hoeft dit laatste niet op te gaan voor de 'aspect ratio', maar het geldt wel voor hoekpuntnummers en assen. Dus als deze in het beeld aanwezig zijn en we wensen ze niet te zien in het eindresultaat, dan moeten we commando *M* in plaats van *E* gebruiken, zoals besproken in paragraaf 1.4. Behalve de mode, kan het ook gewenst zijn het oogpunt te veranderen, waarvoor we commando *V* kunnen gebruiken, zoals is besproken in paragraaf 1.2. Zolang we daar zelf geen verandering in brengen blijft het laatst opgegeven oogpunt in gebruik.

De figuren 1.11(a) en (b) tonen het scherm voor en na het gebruik van commando *E*. Het getoonde voorwerp is een massieve letter L, afgebeeld als draadmodel.

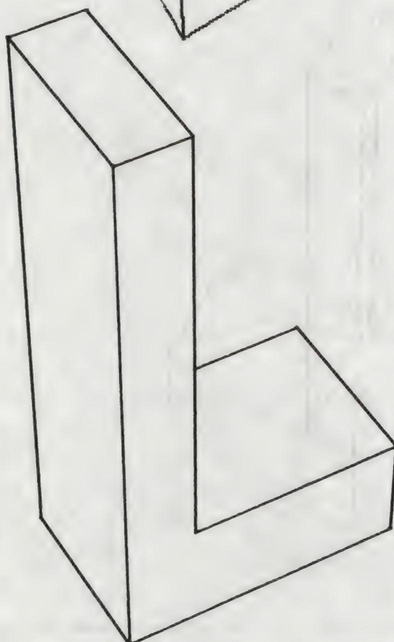
(a)



(b)



(c)



Figuur 1.11 L-vormig voorwerp

(a) op het werkscherm
(b) na commando E
(c) na commando H

In de meeste toepassingen geven we de voorkeur aan *massieve modellen* boven draadmodellen, met andere woorden, we willen dat *verborgen lijnen* worden geëlimineerd. Programma D3D is hiertoe in staat. Wanneer het voorwerp op het scherm afgebeeld is als draadmodel kunnen we typen

```
H
```

(als afkorting van 'Hidden-line elimination'). De volgende vraag verschijnt dan op het scherm:

```
Wilt u de grensvlakken gekromde oppervlakken laten benaderen? (J/N):
```

Laten we voorlopig deze vraag ontkennend beantwoorden en *N* typen. We kunnen vervolgens kiezen uit twee mogelijkheden, die we al bij de bespreking van commando *E* hebben leren kennen:

A aanbevolen in combinatie met commando *P* voor uitvoer op een matrixprinter;

O om uitvoer in een plotfile te verkrijgen.

Als we geen van beide wensen dan kunnen we een willekeurige andere toets indrukken. Hierna kunnen we commando *P* geven om uitvoer op de matrixprinter te verkrijgen. Afgezien van de vraag betreffende gekromde oppervlakken heeft commando *H* dezelfde keuzemogelijkheden als commando *E*. Ook nu kunnen we met *A* te kennen geven dat de 'aspect ratio' op de printer goed moet zijn en dat we daarvoor incorrecte verhoudingen op het scherm op de koop toe nemen en ook nu kunnen we door middel *O* opgeven dat we een plotfile als bijprodukt willen hebben.

Het belangrijkste van commando *H* is uiteraard dat nu de eliminatie van verborgen lijnen plaatsvindt. Bij een gecompliceerd voorwerp (en bij gebruik van een relatief langzame PC met een 8088-processor) kan het enige tijd duren voordat het perspectivische beeld verschijnt, zodat we wat geduld moeten oefenen. Als we ons laatste voorbeeld gebruiken verkrijgen we nu figuur 1.11(c). Het meest interessant in deze figuur is misschien wel de horizontale ribbe (ofwel 'rand') die gedeeltelijk zichtbaar en gedeeltelijk onzichtbaar is.

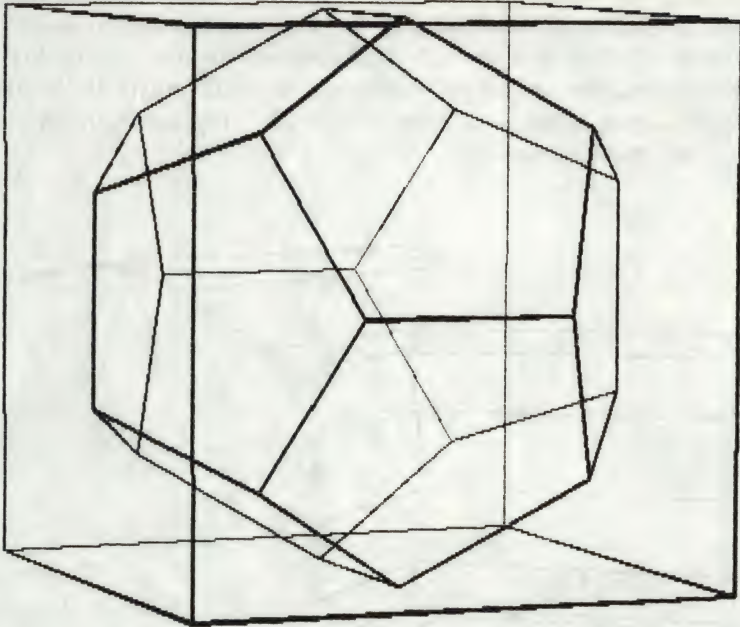
Op commando *H* is het mode-commando *M* niet van toepassing: het eindresultaat gebruikt altijd het gehele scherm en vertoont geen puntnummers, assen, hulplijnen en verschil in lijndikte. Als we (door een willekeurige toets in te drukken) terugkeren naar het normale werkscherm dan wordt de oude 'mode' weer in ere hersteld.

Figuur 1.11(c) is niet op de matrixprinter maar op een HP-plotter vervaardigd, waarbij gebruik is gemaakt van de optie *O* en van programma PLOTHP (zie paragraaf 4.5). De andere en meer directe uitvoermogelijkheid met behulp van commando *P* kunnen we niet alleen na *E* of *H* maar ook direct vanaf het werkscherm gebruiken, zoals figuur 1.11(a) demonstreert; aangezien dit laatste in de regel minder interessant is komt *P* niet voor in het getoonde commandomenu. Uitvoer op de plotter door middel van programma PLOTHP geeft geen puntnummers of lijnen van verschillende dikte. Dit verklaart dat niet alle illustraties in dit boek op dezelfde manier zijn verkregen. Enerzijds wilde ik de kwaliteit zo goed mogelijk hebben, zodat ik de plotter heb gebruikt waar dat maar mogelijk was, maar aan de andere kant had ik in sommige gevallen er behoefte aan puntnummers te tonen en lijnen van verschillende dikte te gebruiken en in deze laatste gevallen heb ik gebruik gemaakt van de matrixprinter. Let wel, het kan heel goed zijn dat u voor uw toepassingen niet echt een plotter nodig heeft, want u kunt tekeningen zoals figuur 1.11(c) ook verkrijgen door gebruik te maken van een matrixprinter. Als u ervoor zorgt dat er nieuw lint in de printer zit dan zal de kwaliteit van het resultaat voor veel toepassingen voldoende zijn.

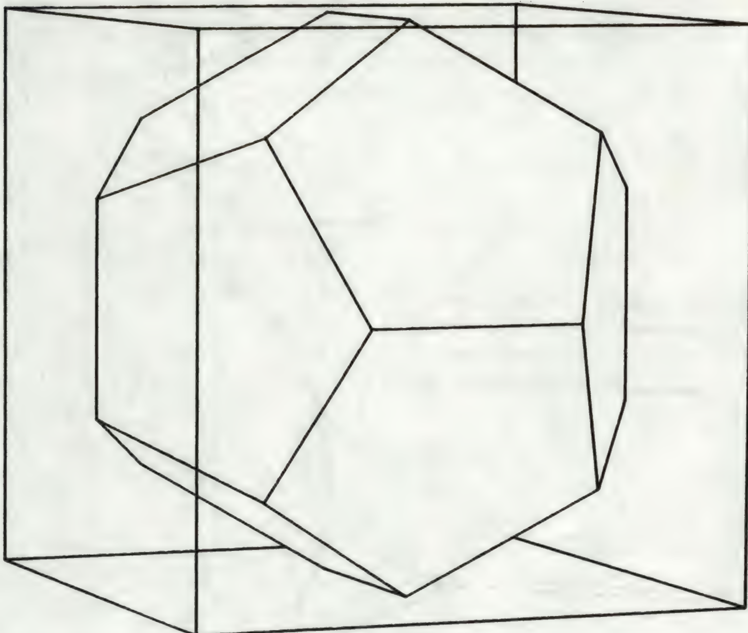
We keren nu terug tot ons eigenlijke onderwerp, de commando's *E* en *H,* waarmee ook de figuren 1.12(a) en (b) zijn verkregen. Het getoonde voorwerp is een regelmatig twaalfvlak (een *dodecaëder*) die netjes in een kubus past. Dit voorwerp kan worden verkregen door commando *R* (in het kort besproken in paragraaf 1.1) toe te passen op de file EXAMPLE2.DAT die zich bevindt op één van de door de uitgever geleverde diskettes. De grensvlakken van een regelmatig twaalfvlak zijn alle regelmatige vijfhoeken, zoals we uitvoerig zullen bespreken in paragraaf 3.5.4.

Figuur 1.12(a) geeft het twaalfvlak weer als draadmodel, terwijl we het in figuur 1.12(b) zien als massief lichaam. De ribben van de kubus zijn 'losse' lijnstukken. Zoals we in paragraaf 1.4 hebben besproken, kunnen we zulke lijnstukken één voor één invoeren met behulp van commando *F,* bij wijze van spreken als grensvlakken met maar twee hoekpunten. Op deze wijze bereiken we dat onze kubus niet anders dan als een draadmodel kan worden afgebeeld: er zijn geen grensvlakken die iets kunnen verbergen. Anderzijds kunnen de ribben van de kubus wel ten dele onzichtbaar gemaakt worden door het twaalfvlak, zoals figuur 1.12(b) laat zien. Ook nu zijn lijnstukken die gedeeltelijk zichtbaar een gedeeltelijk onzichtbaar zijn het interessantst: zij laten duidelijk zien dat we te doen hebben met een ruimtelijk voorwerp. Hoewel misschien wat minder esthetisch dan figuur 1.12(b), is het draadmodel van het twaalfvlak in figuur 1.12(a) overigens ook redelijk duidelijk, wat te danken is aan het verschil in lijndikte.

(a)



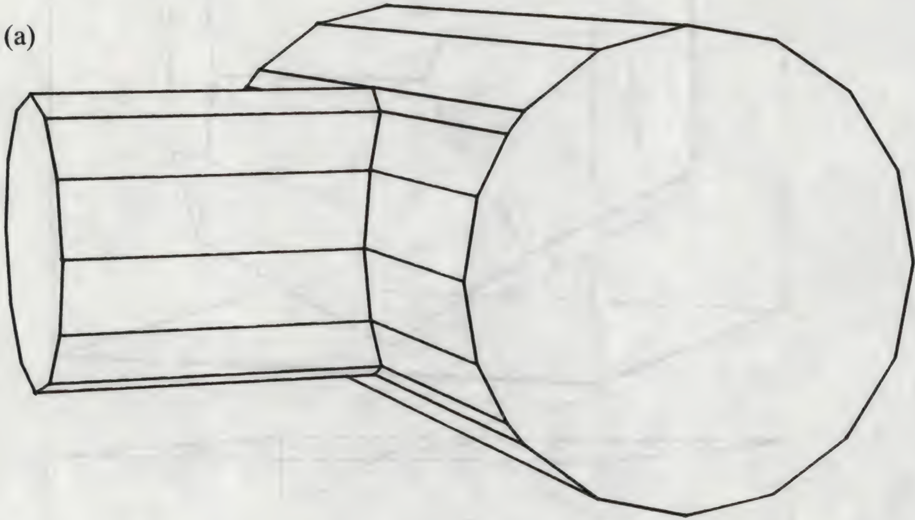
(b)



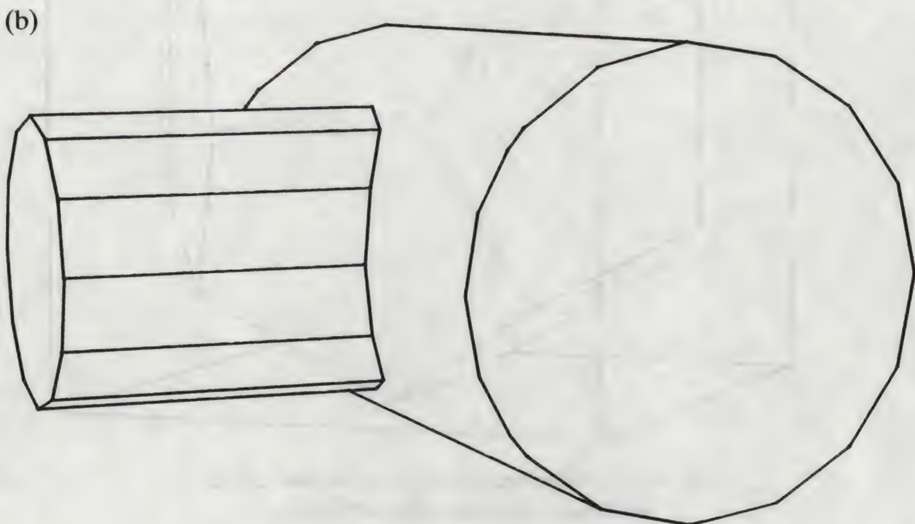
Figuur 1.12 Regelmatig twaalfvlak in een kubus
(a) Draadmodel (b) Massief model

We komen nu terug op de eerder genoemde vraag betreffende gekromde oppervlakken, die onmiddellijk na commando *H* op het scherm verschijnt. Voor de bespreking van dit onderwerp bekijken we de figuren 1.13(a), (b) en (c), die vervaardigd kunnen worden met behulp van de file EXAMPLE3.DAT, die staat op de bij dit boek verkrijgbare diskette. Paragraaf 3.11 zal overigens laten zien hoe we zelf zo'n file kunnen genereren.

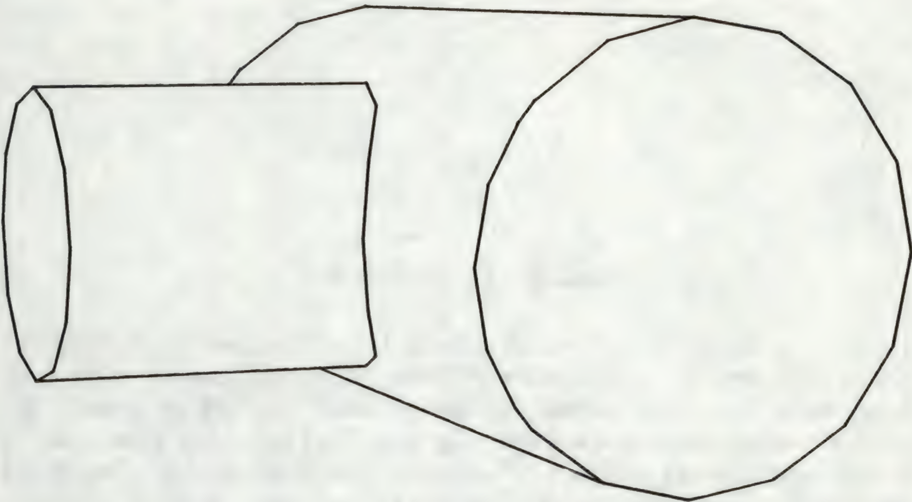
(a)



(b)



(c)



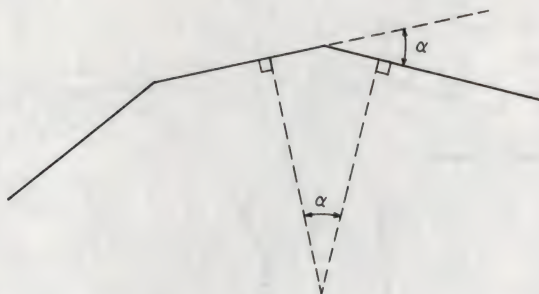
Figuur 1.13 Snijdende cilinders

(a) $\alpha = 0°$; *(b)* $\alpha = 25°$; *(c)* $\alpha = 35°$

Het getoonde voorwerp, een 'T-stuk', suggereert dat het is samengesteld uit twee cilinders. In werkelijkheid accepteert D3D alleen platte grensvlakken, maar door het aantal daarvan groot genoeg te kiezen kunnen we gekromde oppervlakken goed benaderen. In figuur 1.13(a) zien we (voor zover ze zichtbaar zijn) duidelijk alle grensvlakken. Alle zichtbare snijlijnen zijn erin afgebeeld als getrokken lijnen. De meeste ervan zijn lijnen die op een cilinder liggen en evenwijdig zijn met de as van die cilinder. Zij komen alleen maar voort uit de gebruikte benaderingsmethode en zij bestaan niet als snijlijnen op echte cilinders. In figuur 1.13(c) zijn deze lijnen weggelaten. We kunnen dit bereiken door *J* te typen in antwoord op de vraag

```
Wilt u de grensvlakken gekromde oppervlakken laten
benaderen? (J/N)
```

en door een geschikte 'drempelwaarde' α op te geven die nu wordt gevraagd. Figuur 1.14 laat de betekenis van hoek α zien.



Figuur 1.14 Drempelwaarde α

Hoek α wordt gebruikt om er andere hoeken mee te vergelijken, namelijk die tussen elk tweetal aangrenzende vlakken, of liever gezegd, tussen de normaalvectoren op deze vlakken. Als die hoek kleiner dan of gelijk is aan α, dan wordt de snijlijn van de twee vlakken weggelaten; is hij groter dan α dan wordt de snijlijn, voor zover hij zichtbaar is, getrokken. De drempelwaarde α wordt dus gebruikt om te beslissen welke (zichtbare) ribben van het lichaam getekend moeten worden. Het vergroten van α zal vaak het aantal getekende ribben verkleinen. Als α= 0° zullen alle ribben worden getekend, op dezelfde manier als wanneer we *N* antwoorden op bovenstaande vraag betreffende gekromde oppervlakken. Let erop dat de drempelwaarde α alleen van toepassing is op zichtbare lijnen en niets te maken heeft met verborgen lijnen. De laatste worden na commando *H* nooit getekend, ongeacht de waarde α. Uiteraard kan een waarde van α groter dan nul de rekentijd gunstig beïnvloeden, omdat het aantal lijnstukken dat wellicht getekend zal moeten worden erdoor vermindert, hetgeen ook het rekenwerk voor het weglaten van verborgen lijnen reduceert. We kunnen de file EXAMPLE3.DAT gebruiken om eens wat met verschillende waarden van te experimenteren. Gebruiken we α= 25°,dan worden de evenwijdige lijnen op de kleinste cilinder getekend, terwijl die op de grootste cilinder worden weggelaten, zoals figuur 1.13(b) laat zien. Bij α= 35° worden die lijnen op beide cilinders weggelaten, zie figuur 1.13(c). Tussen twee haakjes, deze drempelwaarde van 35° wordt als 'default' genomen als we voor α geen getal opgeven maar in plaats daarvan alleen maar op de Enter-toets drukken.

1.6 OBJECTFILES

In paragraaf 1.2 hebben we gezien dat commando

R

gebruikt kan worden om een file zoals EXAMPLE.DAT in te lezen. Omdat we erg vaak programma D3D zullen willen starten om meteen hierna commando *R* te

gebruiken, is er een iets gemakkelijker middel om hetzelfde te bereiken. We kunnen de naam van de file die gelezen moet worden ook opgeven in de commandoregel, zoals bijvoorbeeld in

```
D3D EXAMPLE1.DAT
```

Op deze manier is het niet meer nodig vlak hierna commando R te gebruiken. Als u (net als ik) een computer met een 8088-processor gebruikt dan zult u merken dat het tekenen van lijnen in verschillende dikten aanzienlijk langzamer gaat dan het trekken van alleen dunne lijnen. Dit kan vervelend zijn, vooral als we moeten wachten op een beeld dat we daarna toch weer gaan veranderen, bijvoorbeeld door commando *H* te gebruiken. Daarom krijgen we, als we D3D op de boven aangegeven manier starten, de gelegenheid om op te geven dat alle lijnen even dun moeten worden. In dat geval typen we namelijk *N* in antwoord op de volgende vraag die op het beeldscherm verschijnt:

```
Wenst u verschil in lijndikte? (J/N):
```

Deze vraag heeft dezelfde betekenis als die over 'Dik en dun', die zoals we aan het eind van paragraaf 1.4 hebben besproken, verschijnt na commando *M*.

De nieuwe manier om een invoerfile op te geven maakt commando *R* niet overbodig, want we kunnen dit laatste goed gebruiken als we een file willen lezen als er al iets op het scherm wordt afgebeeld. In dat geval verschijnt de volgende vraag:

```
Vervangen? (J/N)
```

Om deze vraag te begrijpen moeten we ons realiseren dat een van de volgende twee mogelijkheden gewenst kan zijn:

1 Het scherm moet worden schoongemaakt, zodat het nieuwe voorwerp dat van de file wordt gelezen het oude vervangt. In dit geval is ons antwoord *J*.

2 Het voorwerp dat op het scherm is afgebeeld moet behouden blijven en worden aangevuld met datgene wat van de file zal worden gelezen. Een eventueel conflict tussen puntnummers moet worden opgelost door de punten die gelezen worden om te nummeren zodat in feite een samengesteld voorwerp wordt geconstrueerd. Als we dit willen is ons antwoord op bovenstaande vraag *N*.

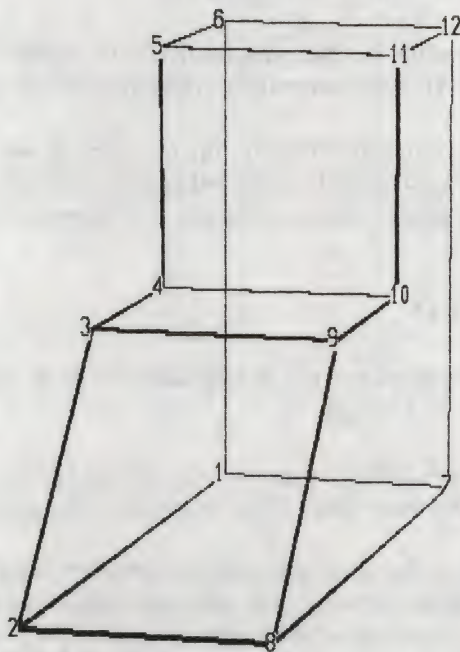
De laatste mogelijkheid biedt ons een middel om een bibliotheek te gebruiken waarin voorwerpen gecodeerd zijn die we vaak in combinatie met andere nodig hebben; we kunnen hier overigens pas echt nuttig gebruik van maken als we vertrouwd zijn met transformaties, die in paragraaf 1.8 worden besproken.

Als we bezig zijn geweest een perspectivisch beeld van een voorwerp te construeren dan kunnen we dat voorwerp opslaan in een objectfile die dezelfde structuur heeft als die welke we als invoer gebruiken. We schrijven het huidige voorwerp (zichtbaar op het scherm) in een objectfile door middel van commando *Write*, afgekort als

```
W
```

Files die met *W* geschreven zijn kunnen met *R* worden gelezen.

Laten we opnieuw het voorwerp gecodeerd in de file EXAMPLE1.DAT gebruiken om de structuur van onze objectfiles te bespreken. Figuur 1.15 toont dat voorwerp met de erbij behorende puntnummers. De assen zijn niet getekend omdat we ons die gemakkelijk kunnen voorstellen: punt 1 is de oorsprong en de punten 2, 7 en 6 liggen respectievelijk op de x-, de y- en de z-as.



Figuur 1.15 Voorwerp en puntnummering

Als we niet met programma D3D bezig zijn, kunnen we de inhoud van de file EXAMPLE1.DAT op verschillende manieren zichtbaar maken, zoals bijvoorbeeld door te typen

```
TYPE EXAMPLE1.DAT
```

of

```
TYPE EXAMPLE1.DAT >LPT1
```

als we een afdruk ervan willen hebben. We verkrijgen dan de lijst met getallen die getoond wordt in tabel 1.1.

Tabel 1.1 Inhoud van file EXAMPLE1.DAT

```
1 0.000000 0.000000 0.000000
2 3.000000 0.000000 0.000000
3 2.000000 0.000000 1.000000
4 1.000000 0.000000 1.000000
5 1.000000 0.000000 2.000000
6 0.000000 0.000000 2.000000
7 0.000000 1.000000 0.000000
8 3.000000 1.000000 0.000000
9 2.000000 1.000000 1.000000
10 1.000000 1.000000 1.000000
11 1.000000 1.000000 2.000000
12 0.000000 1.000000 2.000000

Faces:
 1 2 3 4 5 6.
 7 12 11 10 9 8.
 2 8 9 3.
 3 9 10 4.
 4 10 11 5.
 5 11 12 6.
 1 6 12 7.
 2 1 7 8.
```

Als u mijn boek *Programming Principles in Computer Graphics* hebt gelezen dan zult u opmerken dat deze file dezelfde structuur heeft als soortgelijke files in dat boek, met uitzondering van het begin: in genoemd boek beginnen de objectfiles met de drie coördinaten van een min of meer centraal in het voorwerp gelegen punt. Deze zijn niet langer nodig (en mogen ook niet aanwezig zijn) omdat D3D ze zelf berekent. (Het karakter # is nu vervangen door een punt, maar het mag desgewenst nog steeds in plaats hiervan worden gebruikt.) Laten we nu bekijken hoe een objectfile een voorwerp beschrijft. Tabel 1.1 begint met twaalf regels, elk bestaande uit vier getallen: een positief geheel getal dat een hoekpuntnummer voorstelt

en dat ook in figuur 1.15 te zien is, gevolgd door de drie coördinaten x, y, z. Na deze beschrijving van alle hoekpunten komen we bij het tweede deel van de file, dat begint met de regel

```
Faces:
```

Elk grensvlak wordt nu gespecificeerd door middel van zijn hoekpunten, en wel in een volgorde die we verkrijgen als we de hoekpunten antikloksgewijs doorlopen, waarbij we van de buitenkant tegen het grensvlak aankijken. Het laatste puntnummer van elk grensvlak wordt onmiddellijk gevolgd door een punt. Elk grensvlak is een willekeurige veelhoek. Nu kan het voorkomen dat een veelhoek zoveel hoekpunten heeft dat hun nummers niet op een regel passen en dus over verschillende regels verdeeld moeten worden. We kunnen daarom niet het eind van de regel gebruiken als een signaal dat we het laatste hoekpunt van de veelhoek gehad hebben. Het doel van de punt na het laatste hoekpuntnummer van elk grensvlak zal hiermee duidelijk zijn. Als deze punt door slechts twee puntnummers wordt voorafgegaan, dan stellen die geen veelhoek maar een los lijnstuk voor.

1.7 CONCAVE HOEKPUNTEN; GATEN

Bij de meeste hoekpunten van de gebruikte veelhoeken is de inwendige hoek kleiner dan 180°. We noemen zulke hoekpunten *convex*. Als alle hoekpunten van een veelhoek convex zijn, dan zeggen we dat de veelhoek zelf convex is. Een hoekpunt heet *concaaf* als zijn inwendige hoek groter is dan 180°. Dit is het geval met hoekpunt 2 in figuur 1.16(a), dat het vooraanzicht is van de massieve letter V afgebeeld in figuur 1.16(b). Zoals we weten, wordt een grensvlak van een voorwerp gespecificeerd als een rij hoekpuntnummers, die correspondeert met het antikloksgewijs doorlopen van alle hoekpunten. Programma D3D (of meer in het bijzonder commando *H*) gebruikt de oriëntatie van de eerste drie hoekpunten van zulke rijen om (potentieel) zichtbare grensvlakken te onderscheiden van achtervlakken. Zo kunnen we bijvoorbeeld de veelhoek van figuur 1.16(a) specificeren als

```
3 1 6 5 2 4.
```

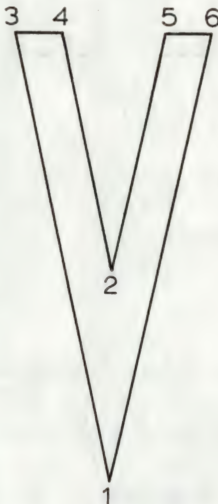
Omdat de eerste drie hoekpunten (3, 1, 6) anti-kloksgewijs worden doorlopen is deze veelhoek potentieel zichtbaar. Het zou daarentegen verwarrend zijn als we de rij

```
5 2 4 3 1 6.
```

opgaven. Nu corresponderen de eerste drie hoekpunten (5, 2, 4) met kloksgewijze oriëntatie, waardoor D3D de veelhoek voor een achtervlak zou houden. De

oorzaak van de verwarring ligt in de ongelukkige keuze van een concaaf hoekpunt (met nummer 2) in de tweede positie van de rij. We zullen dit daarom verbieden en eisen dat het tweede puntnummer overeenkomt met een convex hoekpunt.

(a)

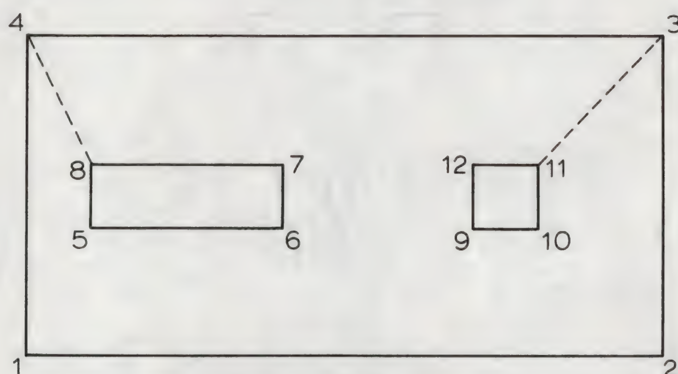


(b)



Figuur 1.16 (a) Veelhoek met een concaaf hoekpunt
(b) Resultaat van commando H

Tot dusverre zijn we steeds bezig geweest met echte veelhoeken, dat wil zeggen we hebben aangenomen dat het volledige inwendige deel van elke veelhoek een grensvlak van het beschouwde voorwerp is. We stappen nu over op veelhoeken waarin gaten zitten en we zullen daar een speciale behandeling op toepassen. Neem bijvoorbeeld de rechthoek met twee gaten, afgebeeld in figuur 1.17.



Figuur 1.17 Een veelhoek met gaten

Dit is geen echte veelhoek, maar we zullen er een veelhoek van maken. Daartoe brengen we twee hulplijnen aan, namelijk 3–11 en 4–8, zoals door middel van stippellijnen aangegeven in figuur 1.17. We kunnen nu het standpunt verdedigen dat de getallenrij

```
1 2 3 11 10 9 12 11 3 4 8 7 6 5 8 4
```

een veelhoek voorstelt, waarin de twee zijden 3–11 en 11–3 samenvallen (evenals de zijden 4–8 en 8–4). We moeten alleen voorkomen dat deze stippellijnen getekend worden, dus op een of andere manier moeten we aangeven dat de twee paren samenvallende zijden in ons voorbeeld volkomen kunstmatig zijn. We doen dit door af te spreken dat we in plaats van het getallenpaar

P Q

het paar getallen

P –Q

gebruiken als PQ zo'n kunstmatige zijde is. Het minteken zal dan voorkomen dat lijnstuk PQ wordt getekend. In ons voorbeeld van figuur 1.17 zal de objectfile daarom de regel

```
1 2 3 -11 10 9 12 11 -3 4 -8 7 6 5 8 -4.
```

kunnen bevatten in plaats van de eerder getoonde regel van positieve getallen. Als we alle hoekpunten in figuur 1.17 in deze volgorde doorlopen, steeds met onze ogen gericht op het volgende hoekpunt, dan is steeds het 'materiaal' van de veelhoek die we beschrijven links van ons. Voor een eenvoudige convexe veelhoek, zoals een rechthoek, betekent dit dat we hem anti-kloksgewijs doorlopen, maar binnen in een gat volgen we de hoekpunten kloksgewijs. Dit verklaart bijvoorbeeld de bovenstaande deelrij

```
10 9 12 11
```

die we niet mogen vervangen door

```
12 9 10 11
```

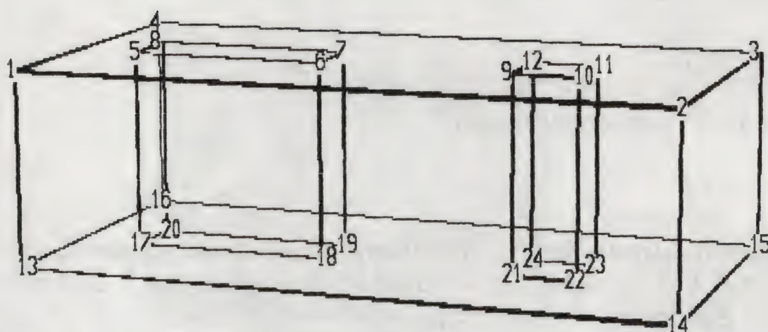
We hebben om twee redenen veel aandacht besteed aan de structuur van objectfiles. Ten eerste kunnen objectfiles, zoals we in de hoofdstukken 2 en 3 zullen zien, worden gegenereerd door andere programma's en behalve gebruik te maken van kant en klare programma's die daar worden besproken, zult u misschien zelf zulke programma's willen schrijven. In de tweede plaats kunnen we, nu we begrijpen hoe veelhoeken met gaten door getallenrijen worden voorgesteld, nog even verder gaan met onze bespreking van commando *F*. We kunnen namelijk ook hierbij negatieve hoekpuntnummers opgeven, met dezelfde betekenis als in objectfiles. Laten we bijvoorbeeld aannemen dat figuur 1.17 het bovenaanzicht is van een voorwerp dat bestaat uit een prisma met twee gaten erin. Het onderaanzicht van het voorwerp is analoog aan het bovenaanzicht en de hoekpuntnummers van het ondervlak worden verkregen door de corresponderende hoekpuntnummers van het bovenvlak met 12 te verhogen. De figuren 1.18(a) t/m (d) tonen dit voorwerp.

We zullen nu, bij wijze van uitzondering, heel gedetailleerd bespreken hoe figuur 1.18(a) t/m (d) kunnen worden verkregen. Alle hoekpunten zullen we definiëren met behulp van de cursor. Het indrukken van de Enter-toets wordt aangegeven met behulp van <Enter>; spaties tussen getallen moeten echt worden ingetypt, andere spaties niet. Commentaar, dat niet wordt ingetypt, staat tussen accolades {}. Zoals u zich zult herinneren, mogen we kleine letters in plaats van hoofdletters gebruiken. De volgende opsomming lijkt veel ingewikkelder dan hij is omdat kortheidshalve is weggelaten wat gedurende dit proces op het scherm te zien is. Zo moge de regel

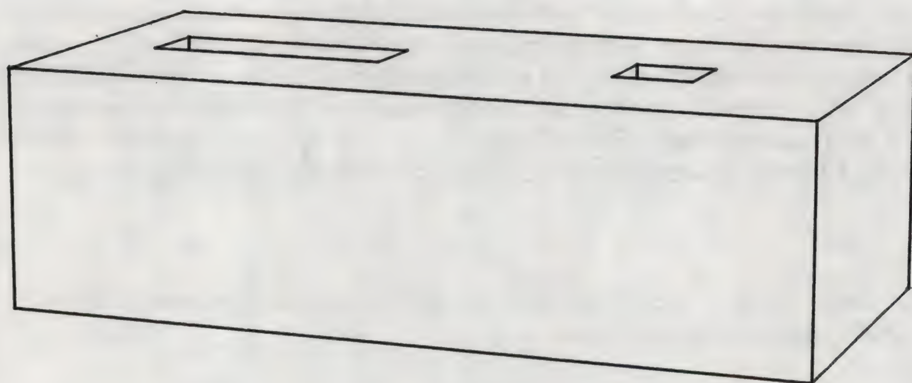
```
HNAP
```

die hieronder twee keer voorkomt er afschrikwekkend uit zien, maar D3D helpt ons als volgt aan deze vier letters.

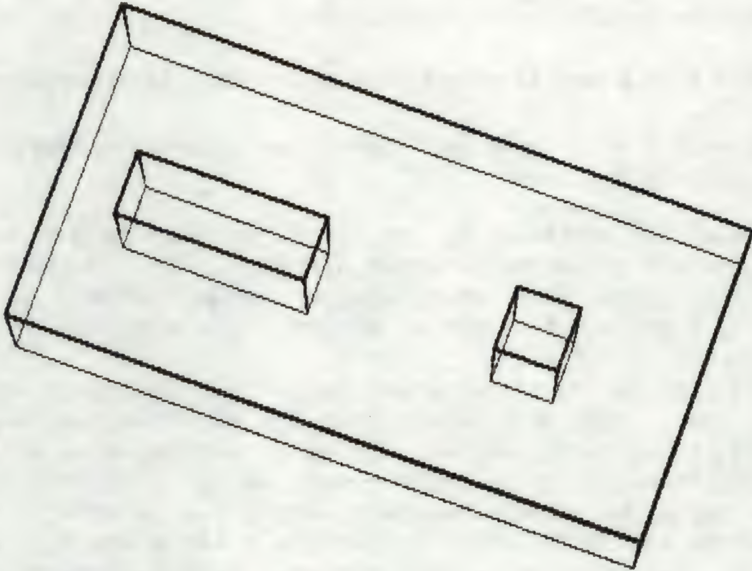
(a)



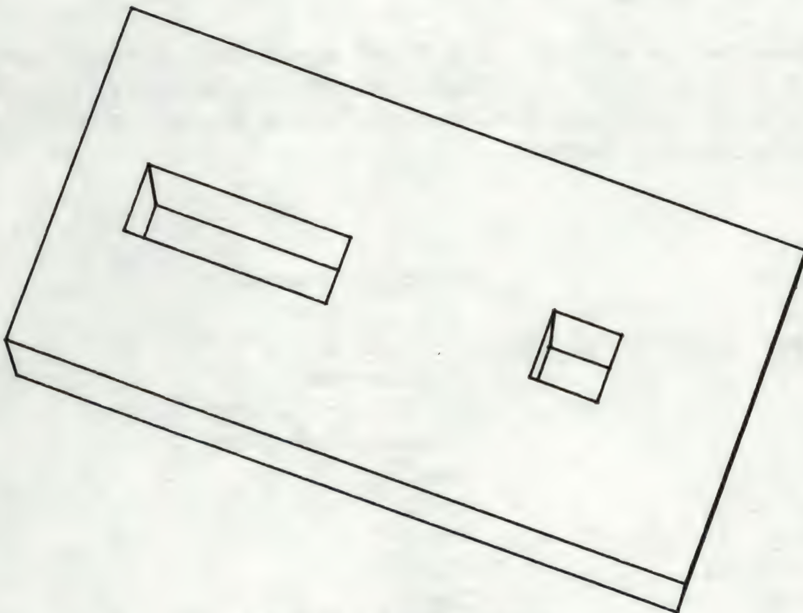
(b)



(c)



(d)



Figuur 1.18 Prisma met gaten

(a) $\varphi = 80°$*, draadmodel; (b)* $\varphi = 80°$*, massief model*
(c) $\varphi = 10°$*, draadmodel; (d)* $\varphi = 10°$*, massief model*

De letter *H* is het afgekorte commando *Hidden*, dat zichtbaar is in het commandomenu. Als we het intypen verschijnt

```
Wilt u de grensvlakken gekromde oppervlakken laten benaderen? (J/N)
```

hetgeen *N*, de tweede letter van *HNAP* verklaart. Daarna verschijnt het volgende stukje tekst op het scherm:

```
Wanneer het resultaat op het scherm zichtbaar is kunt u het met
commando P op een matrixprinter afdrukken. (Als u in plaats hiervan
op een andere toets drukt komt het vorige grafische scherm weer
terug.) Ten aanzien van de printer dient u op het volgende te
letten:
We zeggen dat de 'aspectverhouding' correct is als een cirkel als
een echte cirkel wordt afgebeeld en niet als een ellips. Normaal zal
de aspectverhouding correct zijn op het scherm. Als u een correcte
aspectverhouding op de printer wenst (en een incorrecte aspectver-
houding op het scherm accepteert) druk dan op de toets A. In plaats
hiervan kunt u de letter O typen als u een uitvoerfile (xxx.PLT)
wenst, die bijvoorbeeld door programma PLOTHP kan worden gelezen om
uitvoer op een HP-plotter te verkrijgen. Als geen van beide gewenst
wordt, druk dan op een andere toets ...
```

De hierin voorkomende letters *A* en *P* verklaren de derde en vierde letter van *HNAP*. Nadat we *D3D* hebben ingetypt om het programma te starten, zijn de meeste andere regels hieronder niet moeilijk te bedenken, aangezien op het scherm meestal toelichting te vinden is.

```
D3D
<Enter>                          {Start het programma}
<Enter>
V 10000 <Enter>     0 <Enter>     0 <Enter>
                                 {Bovenaanzicht:
                                 ρ = 10000, θ = 0°, φ = 0°}
Z+++                             {Ga naar het bovenvlak}
X+++++I                          {Punt 1}
Y++++++++++I                     {Punt 2}
X-----I                          {Punt 3}
Y----------I                     {Punt 4}
X+++Y+I                          {Punt 5}
+++I                             {Punt 6}
X-I                              {Punt 7}
Y---I                            {Punt 8}
```

```
++++++X+I                    {Punt 9}
Y+I                          {Punt 10}
X-I                          {Punt 11}
Y-I                          {Punt 12}
F                            {Definieer bovenvlak:}

1 2 3 -11 10 9 12<Enter>
11 -3 4 -8 7 6<Enter>
5 8 -4.<Enter>
V <Enter> <Enter>180<Enter>
                             {Onderaanzicht:
                             ρ = 10000, θ = 0°, φ = 180°}
```

{We zullen nu precies hetzelfde intypen als we hierboven hebben gedaan, te beginnen bij *X+++++I* en eindigend op *F*; ieder bovengenoemd puntnummer *j* komt overeen met een puntnummer *j* + 12 hieronder:}

```
X+++++I                      {Punt 13}
Y++++++++++I                 {Punt 14}
X-----I                      {Punt 15}
Y----------I                 {Punt 16}
X+++Y                        {Punt 17}
+++I                         {Punt 18}
X-I                          {Punt 19}
Y---I                        {Punt 20}
++++++X+I                    {Punt 21}
Y+I                          {Punt 22}
X-I                          {Punt 23}
Y-I                          {Punt 24}
F                            {Definieer ondervlak:}
16 15 14 -22 23 24 <Enter>
21 22 -14 13 -17 <Enter>
18 19 20 17 -13.<Enter>
V 8<Enter>    20<Enter>      80<Enter>
                             {Perspectief, ρ = 8, θ = 20°, φ = 80°}
F                            {Definieer alle verticale grensvlakken}
13 14 2 1.<Enter>            {Buitenkant}
14 15 3 2.<Enter>
15 16 4 3.<Enter>
16 13 1 4.<Enter>

18 17 5 6.<Enter>            {Linker gat}
17 20 8 5.<Enter>
20 19 7 8.<Enter>
```

```
19 18 6 7.<Enter>

22 21 9 10.<Enter>                 {Rechter gat}
21 24 12 9.<Enter>
24 23 11 12.<Enter>
23 22 10 11.<Enter>

MJNJJ                              {Mode; Puntnummers? Ja.
                                   Assen? Nee. Dik en dun? Ja.
                                   Geheel scherm? Ja.}

AP                                 {Aspect-ratio, Print figuur 1.18(a)}
HNAP                               {Hidden. Geen gekromde oppervlakken.
                                   Aspect. Print figuur 1.18(b).}
MNNJN                              {Mode. Geen puntnummers. Geen assen.
                                   Dik en dun. Geen volledig scherm.}
V 20<Enter>     20<Enter>     10<Enter>
                                   {Perspectief, ρ = 20, θ = 20°, φ = 10°}
EAP                                {Print figuur 1.18(c)}
HNAP                               {Print figuur 1.18(d)}
W                                  {Schrijf objectfile}
HOLES.DAT <Enter>                  {Naam van objectfile}
Q                                  {Quit}
```

Ongetwijfeld is het invoeren van de grensvlakken het lastigst, omdat we hierbij moeten letten op de oriëntatie. Er zijn enkele regels en trucs die deze taak vergemakkelijken en omdat het kennen ervan D3D tot een bruikbaar programma kan maken, zullen we ze hieronder bespreken:

1 Bij ingewikkelde veelhoeken, vooral als deze, zoals in ons voorbeeld, gaten bevatten, kan het de moeite waard zijn het oogpunt anders te kiezen, zodanig dat we het grensvlak dat we aan het specificeren zijn zien vanaf de buitenkant van het voorwerp. Zo kunnen we bijvoorbeeld φ gelijk aan 180° kiezen als we bezig zijn met de onderkant van een voorwerp.
2 Het moet sterk worden aanbevolen een systematische volgorde aan te houden, zowel bij het opgeven van de grensvlakken als voor de hoekpunten van ieder grensvlak afzonderlijk. In ons voorbeeld hebben we drie groepen van vier rechthoeken en elke keer beginnen we met de dichtstbijzijnde ribbe in het grondvlak.
3 Soms worden de hoekpuntnummers onleesbaar omdat zij elkaar overlappen. In zulke situaties kan commando *E* nuttig zijn; door het gehele scherm te gebruiken wordt het beeld vergroot, waardoor puntnummers weer leesbaar kunnen worden. Het kan ook handig zijn na *E* de tot dusverre ontstane situatie op de printer te laten afdrukken, of, meer primitief, er een schetsje van te

maken waarin ook de puntnummers zijn aangegeven.

4 Als we de ingevoerde grensvlakken willen controleren of corrigeren kunnen we commando *L* gebruiken, zoals besproken is in paragraaf 1.4. Dit is vooral belangrijk als we een typefout hebben gemaakt waardoor een verkeerd grensvlak verschijnt, dat al ons werk schijnt te verknoeien. Commando *L* toont ons achtereenvolgens alle vlakken en biedt ons daarbij de mogelijkheid een verkeerd vlak te verwijderen door *N* te typen in antwoord op de vraag *O.K.? (J/N)*.

5 In plaats van met de hand verschillende grensvlakken op te geven die dezelfde vorm hebben, kunnen we er slechts één definiëren en de andere ervan afleiden met behulp van transformaties, die in de volgende paragraaf worden besproken.

1.8 TRANSFORMATIES

Deze paragraaf gaat over transformaties in de driedimensionale ruimte. We kunnen met D3D een rotatie, een translatie, een vermenigvuldiging (ofwel *schaling*) en een spiegeling uitvoeren. Bij elk van deze vier soorten transformaties kunnen we een voorwerp, of een deel ervan, verplaatsen ('move') of kopiëren ('copy'). Als we een voorwerp verplaatsen dan verschijnt een nieuwe versie ervan maar tegelijkertijd wordt de oude versie verwijderd; in dit geval heeft het nieuwe voorwerp precies dezelfde hoekpuntnummers als het oorspronkelijke. Anderzijds, als we een voorwerp kopiëren, dan blijft de oude versie bestaan: het wordt niet vervangen maar gedupliceerd. Na het kopiëren zijn aan de hierbij ontstane versie van het voorwerp nieuwe hoekpuntnummers toegekend. Voor elk type transformatie typen we

```
T
```

waarna de volgende vraag verschijnt:

```
('Move'/'Copy'? (M/C)
```

We dienen deze vraag te lezen als:

```
Wilt u het voorwerp verplaatsen of dupliceren?
```

In het eerste geval typen we *M*, in het tweede *C*. (Het verschil tussen deze twee mogelijkheden lijkt op wat u misschien kent van een tekstverwerker of editor, waar we stukken tekst naar keuze kunnen verplaatsen of kopiëren.) In beide gevallen verschijnt de volgende regel:

```
Ondergrens: 1
```

We kunnen nu gewoon op de Enter-toets drukken.

In plaats hiervan kunnen we een andere ondergrens intypen, laten we zeggen L. Daarna verschijnt de regel

```
Bovengrens: ...
```

waarin ... het grootste in gebruik zijnde puntnummer is. Ook nu kunnen we die waarde accepteren door op de Enter-toets te drukken, of een andere bovengrens, laten we zeggen U, intypen. De nog te specificeren transformatie zal dan worden toegepast op alle hoekpunten met nummers i die voldoen aan

$$L \leqq i \leqq U.$$

Als we dus twee keer de Enter-toets hebben ingedrukt, waarmee we de voorgestelde waarden voor L en U hebben geaccepteerd, dan wordt het volledige voorwerp getransformeerd. Na het *dupliceren* van een voorwerp, of een deel ervan, wordt het bereik van de nieuw toegekende puntnummers op het scherm getoond. Als bijvoorbeeld het oorspronkelijke voorwerp de hoekpuntnummers 1, 2, ..., 10 had en alle hoekpunten hebben deelgenomen aan de ('kopiërende') transformatie, dan zullen de volgende drie regels op het scherm verschijnen:

```
Nieuwe punten:
11-20
Druk op een toets ...
```

Dit bereik (11-20) verschaft ons nuttige informatie, omdat we vaak niet slechts één transformatie willen uitvoeren, maar na elkaar bijvoorbeeld een translatie, een rotatie en een vermenigvuldiging willen uitvoeren. Het is daarom aan te bevelen een aantekening van dit bereik te maken als we het later nodig kunnen hebben om een onder- en bovengrens voor een volgende transformatie op te geven. Natuurlijk kunnen we met commando *M* ervoor zorgen dat de hoekpuntnummers ook in de beeldfiguur verschijnen, maar bij ingewikkelde voorwerpen zullen ze al gauw elkaar overlappen zodat we ze niet goed kunnen lezen. In zulke gevallen kunnen we, indien nodig, commando *M* beter gebruiken om nummers *weg te laten* in plaats van ze te laten zien, waarbij we liever uitsluitend gebruik maken van het bereik zoals dat op het scherm is af te lezen. Om te voorkomen dat dit meteen wordt overschreven door een andere tekstregel, moeten we nu eerst op een toets drukken; pas daarna gaat D3D verder.

We hebben de weergave van het nieuwe puntbereik nu alvast besproken omdat het een gemeenschappelijk aspect is van alle vier de soorten transformaties. In werkelijkheid verschijnt dat puntbereik pas op het scherm nadat de punten in kwestie zijn getransformeerd, dus het wordt tijd dat we die transformaties zelf gaan bespreken. Nadat we een onder- en een bovengrens voor de hoekpuntnummers

hebben opgegeven, zullen achtereenvolgens één of meer van de volgende vragen worden gesteld:

```
Rotatie? (J/N)
Translatie? (J/N)
Schaling? (J/N)
Spiegeling? (J/N)
```

We kunnen één van deze vier transformaties (rotatie, translatie, schaling, spiegeling) kiezen door *J* te antwoorden. Alleen als we niet met *J* antwoorden verschijnt de volgende vraag; als zelfs de laatste vraag met *N* wordt beantwoord dan wordt geen enkele transformatie uitgevoerd en keren we terug naar het normale commandomenu. We zullen de vier transformaties uitvoerig bespreken in de paragrafen 1.8.1 tot en met 1.8.4.

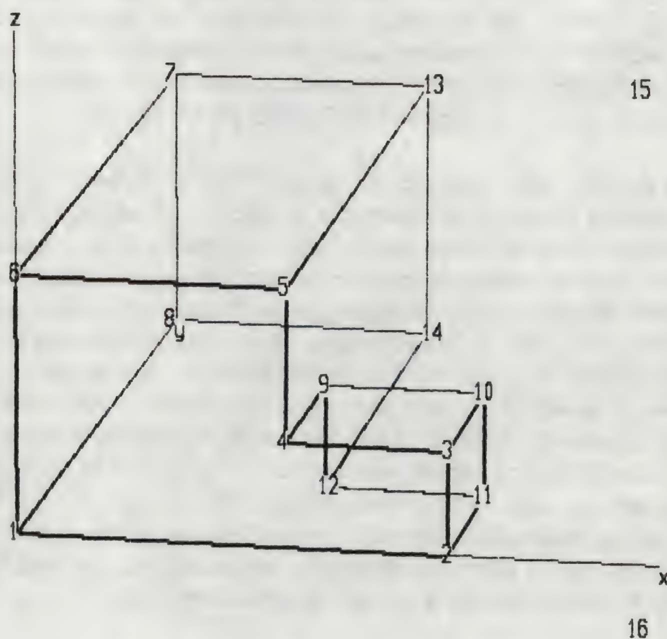
Bij elk van de vier soorten transformaties zal gevraagd worden één of meer puntnummers in te typen bij wijze van meer gedetailleerde informatie omtrent de uit te voeren transformatie. Het zal dan toegestaan zijn het getal 0 te gebruiken om de oorsprong van het coördinatenstelsel aan te duiden. Deze mogelijkheid is in de volgende paragrafen niet expliciet genoemd. Dus als we bijvoorbeeld in paragraaf 1.8.1 worden gewaarschuwd dat we twee punten P en Q moeten definiëren voordat we commando *T* geven, dan hoeven we eigenlijk maar één van deze twee punten vooraf te definiëren als we voor het andere de oorsprong willen nemen. Let erop dat het gebruik van het getal 0 geen verwarring zal veroorzaken omdat alle door ons zelf gedefinieerde punten een nummer hebben dat groter dan 0 is.

Terwille van de volledigheid moeten we nog een andere faciliteit bespreken, die vroeg of laat nuttig zal blijken te zijn en die op alle vier de soorten transformaties van toepassing is. Het gaat ook nu om de puntnummers die we moeten intypen om meer gedetailleerde informatie over een transformatie te verstrekken. Als we deze nummers op de gewone manier, dat wil zeggen als positieve getallen, opgeven, dan nemen de punten deel aan de transformatie. Als we ze daarentegen ervan willen uitsluiten getransformeerd te worden, dan kunnen we dat aangeven door een minteken aan het nummer te laten voorafgaan: we schrijven bijvoorbeeld -25 in plaats van 25. Hoewel de absolute waarde van het ingetypte getal wordt gebruikt als puntnummer, zal D3D zich herinneren dat er een minteken voor heeft gestaan op het moment dat de transformatie wordt uitgevoerd; het punt in kwestie zal dan worden overgeslagen, net als nummers die buiten het opgegeven bereik *L*, *U* liggen. Let wel, deze faciliteit is alleen van toepassing op enkele bijzondere punten, zoals bijvoorbeeld de twee punten die de as van een rotatie bepalen.

1.8.1 Rotatie

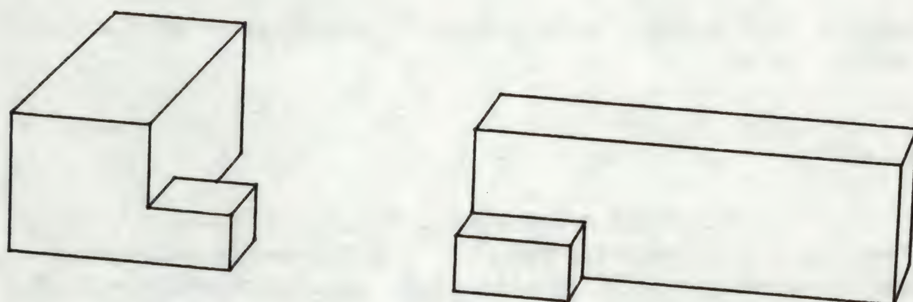
Een rotatie in de driedimensionale ruimte wordt uitgevoerd om een as, waarvan twee punten moeten worden opgegeven; laten we deze punten P en Q noemen. Om de oriëntatie van de rotatie te kunnen vastleggen beschouwen we PQ als een gericht lijnstuk; P en Q kunnen zich overal in de ruimte bevinden, dus de as van rotatie hoeft niet horizontaal of verticaal te zijn. We moeten ook een hoek α opgeven. De rotatie zal dan worden uitgevoerd om de as PQ en over de hoek α, zodanig dat een beweging van P naar Q overeenkomt met de voorwaartse beweging van een rechtse schroef wanneer die gedraaid wordt in de zin van deze rotatie.

De punten P en Q kunnen hoekpunten van het voorwerp zelf zijn, maar in de meeste toepassingen is dat niet wat we willen. Daarom moeten we gewoonlijk de punten P en Q definiëren voordat we commando *T* geven. We zijn niet echt in de problemen als we dat vergeten hebben, want we kunnen dan alle vragen met *N* beantwoorden, of, als we al gezegd hebben dat we willen roteren, een willekeurige as en een hoek van 0° nemen. In het geval van zo'n 'valse start' keren we vanzelf naar het hoofdmenu terug; we definiëren dan (met commando *I* of met gebruik van de cursor) P en Q en geven nogmaals commando *T*.



Figuur 1.19 Te roteren voorwerp en as van rotatie

Beginnend met het voorwerp van figuur 1.19 kunnen we de situatie van figuur 1.20 verkrijgen door het voorwerp te 'kopiëren' in een rotatie die plaatsvindt over een hoek van 90° en om de verticale lijn door de punten 15 en 16 die ook in figuur 1.19 zijn aangegeven.



Figuur 1.20 Resultaat van een rotatie

Laten we aannemen dat we commando *T* (dat wil zeggen *Transform*) hebben gegeven en dat we *C* hebben geantwoord op de vraag

```
'Move'/'Copy' (M/C).
```

Nadat we twee keer op de Enter-toets hebben gedrukt om aan te geven dat alle hoekpunten aan de transformatie moeten deelnemen en nadat we op de zojuist besproken wijze hebben duidelijk gemaakt dat de uit te voeren transformatie een rotatie moet zijn, verschijnen de volgende twee regels op het beeldscherm:

```
Rotatie om PQ.
Puntnrs. P en Q:
```

In ons voorbeeld typen we

```
15 16
```

waarna de op de Enter-toets drukken. Er verschijnt dan als volgt een verzoek om de hoek van rotatie op te geven:

```
Hoek in graden:
```

Als we nu het getal

```
90
```

intypen, dan zal een kopie van het oorspronkelijke voorwerp, geroteerd over 90° om de opgegeven as worden gecreëerd. Als we vervolgens met commando *H* de verborgen lijnen verwijderen, verkrijgen we het resultaat zoals getoond in figuur 1.20.

Let erop dat de punten P en Q in zichzelf worden getransformeerd, zodat we kunnen schrijven:

$$P \equiv P'$$
$$Q \equiv Q'$$

We zeggen dat P en Q (en in feite alle punten op de lijn PQ) *vaste punten* zijn. Aannemende dat we aan het 'kopiëren' zijn, waarbij nieuwe puntnummers aan de gekopieerde punten worden toegekend, kunnen we ons afvragen of we wel willen dat aan P' en Q' nieuwe puntnummers worden toegekend. Als P en Q zelf punten van het getransformeerde voorwerp zijn dan willen we dat meestal inderdaad, omdat op die manier zowel het oorspronkelijke als het nieuwe voorwerp een eigen puntbereik hebben, zodat we een volgende transformatie alleen op het nieuwe voorwerp kunnen toepassen. In figuur 1.19 evenwel behoren de punten 15 en 16 niet tot het voorwerp; in gevallen als deze willen we gewoonlijk niet dat nog eens een nieuw stel nummers aan deze vaste punten wordt toegekend. Programma D3D komt aan deze wens tegemoet door bij het transformeren een uitzondering te maken voor de punten P en Q als we hun nummers bij het intypen laten voorafgaan door een minteken. In ons voorbeeld hadden we dus

```
-15  -16
```

kunnen intypen om dit te bereiken. (Overigens is er in dit voorbeeld een andere manier om hetzelfde te bereiken: we kunnen eenvoudig 14 intypen als bovengrens van alle puntnummers die bij de transformatie betrokken moeten worden, in plaats van door een druk op de Enter-toets de voorgestelde bovengrens, 16, te accepteren.)

1.8.2 Translatie

We zeggen dat we een *translatie* uitvoeren als we een voorwerp verschuiven, zonder daarbij zijn stand te veranderen, zodat alle lijnstukken in het voorwerp hun oorspronkelijke richting behouden. Meer analytisch uitgedrukt, we gebruiken drie constanten *delta x*, *delta y* en *delta z*, en we berekenen

$x' = x + delta\ x$
$y' = y + delta\ y$
$z' = z + delta\ z$

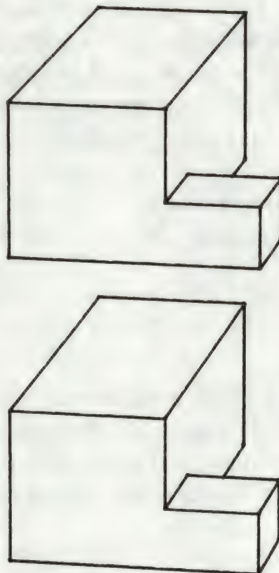
voor elk punt P(x, y, z) van het voorwerp, om aldus het overeenkomstige nieuwe punt P'(x', y', z') te vinden.

In D3D kunnen we òf de drie waarden *delta x*, *delta y*, *delta z* expliciet als drie numerieke constanten opgeven òf gebruik maken van een *schuifvector* AB. Elk gericht lijnstuk AB kan als zo'n schuifvector dienst doen. Na ons antwoord *J* op de vraag *Translatie? (J/N)* verschijnt de volgende vraag:

```
Schuifvector? (J/N)
```

We typen *N* als we de drie waarden *delta x*, *delta y*, *delta z* willen intypen en *J* als we liever een schuifvector opgeven. In het eerstgenoemde geval wordt om die drie waarden gevraagd en in laatstgenoemd geval verschijnen de volgende twee regels op het scherm:

```
AB is schuifvector
Puntnrs. A en B:
```



Figuur 1.21 Resultaat van een translatie

We moeten dan de nummers van twee tevoren gedefinieerde punten intypen. (Let erop dat dit net eender is als bij de rotatie, waarbij we de tevoren gedefinieerde punten P en Q moesten opgeven.) Hoewel we de nummers van de punten A en B opgeven wordt ook nu gebruik gemaakt van de drie waarden *delta x*, *delta y*, *delta z*, die nu als volgt worden berekend:

$$delta\ x = x_B - x_A$$
$$delta\ y = y_B - y_A$$
$$delta\ z = z_B - z_A$$

Figuur 1.21 toont een voorbeeld van een translatie. Ook dit resultaat is gebaseerd op figuur 1.19; de punten 16 en 15 (in die volgorde) zijn gebruikt als de punten P en Q van schuifvector PQ.

Als we aan het 'kopiëren' zijn (dus niet aan het 'verplaatsen'), dan kunnen we, zoals eerder vermeld, voorkomen dat de punten A en B, als zij in het opgegeven nummerbereik liggen, gekopieerd worden, door hun nummers te voorzien van een minteken.

1.8.3 Schaling

Door een voorwerp vanuit een punt met een getal (ongelijk aan 1) te vermenigvuldigen veranderen we het van grootte. In de internationale literatuur gebruikt men hiervoor meestal het woord *scaling* en om deze term zo goed mogelijk te benaderen zullen we spreken over *schaling*. Om expliciet aan te geven dat het schalen in alle richtingen met dezelfde factor gebeurt (als we dat willen) spreken we ook wel over *uniform* schalen. Het getal S waarmee alle afmetingen worden vermenigvuldigd heet de *schaalfactor*. Een generalisatie hiervan is het toepassen van niet noodzakelijk gelijke schaalfactoren S_x, S_y, S_z in de drie richtingen x, y, z van het coördinatenstelsel. Het is duidelijk dat het uniform schalen hiervan een bijzonder geval is, waarbij

$$S_x = S_y = S_z = S$$

Voordat we de schalingsoperatie kunnen uitvoeren moeten we ook het *vaste punt* kennen, dat wil zeggen het punt dat bij het schalen op zijn plaats blijft. Als we voor dit doel de oorsprong van het coördinatenstelsel gebruiken, dan wordt aan elk punt $P(x, y, z)$ een nieuw punt $P'(x', y', z')$ toegekend, waarbij geldt:

$$x' = S_x . x$$
$$y' = S_y . y$$
$$z' = S_z . z$$

Ook dit willen we generaliseren, zodat we ook andere punten als vast punt kunnen kiezen. Als dat vaste punt ('fixed point') F(x_F, y_F, z_F) is, dan gebruiken we $x - x_F$ in plaats van x, $x' - x_F$ in plaats van x', enzovoort, hetgeen leidt tot de volgende schalingsformules:

$$x' - x_F = S_x \cdot (x - x_F)$$
$$y' - y_F = S_y \cdot (y - y_F)$$
$$z' - z_F = S_z \cdot (z - z_F)$$

We kunnen deze in de volgende vorm brengen, wat de efficiency ten goede komt:

$$x' = S_x \cdot x + C_1$$
$$y' = S_y \cdot y + C_2$$
$$z' = S_z \cdot z + C_3$$

waarin C_1, C_2, C_3 constanten zijn, die niet van x, y, z afhangen en daarom alleen berekend hoeven te worden voordat het eigenlijke schalen plaatsvindt:

$$C_1 = x_F - S_x \cdot x_F$$
$$C_2 = y_F - S_y \cdot y_F$$
$$C_3 = z_F - S_z \cdot z_F$$

Voor ons als gebruiker gaat het schalen als volgt. Na commando *T* beantwoorden we de vraag *Schaling? (J/N)* met *J*. Er verschijnt dan de volgende regel op het scherm:

```
Uniform? (J/N)
```

Als ons antwoord hierop *J* is, dan verschijnt de regel

```
Sx=Sy=Sz=
```

waarna we de verlangde uniforme schaalfactor intypen. Antwoorden we *N* op bovenstaande vraag, dan verschijnen achtereenvolgens de drie regels

```
Sx =
Sy =
Sz =
```

zodat we in elk van de drie hoofdrichtingen de toe te passen schaalfactor kunnen intypen. Hierna komen de volgende drie regels op het scherm te staan:

```
Vast punt:
Centr. (C), Oorsprong (O)
of een Hoekpunt (H):
```

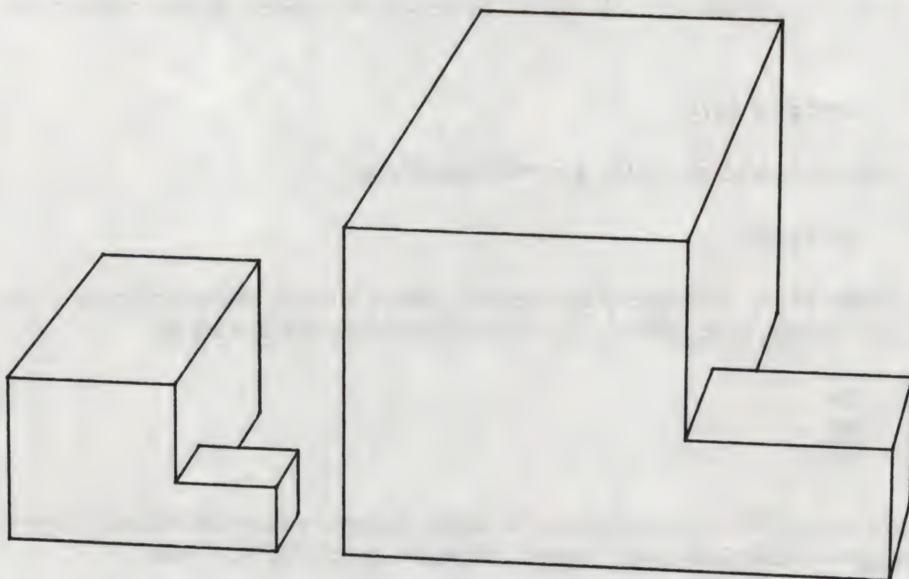
We moeten nu een van de drie letters *C*, *O* en *H* intypen. Als we O intypen dan is de oorsprong van het coördinatenstelsel het vaste punt. Typen we

```
H
```

dan wordt als volgt gevraagd een puntnummer in te typen.

```
Puntnummer:
```

De derde mogelijkheid, *C*, betekent dat het centrum van het voorwerp als vast punt wordt gekozen. Als er een assenstelsel op het scherm zichtbaar is, dan zal dit ook meedoen bij de bepaling van het centrum, zodat het berekende vaste punt iets kan verschillen van dat wat we verwachten. Als de keuze van het vaste punt erg belangrijk is, dan kunnen we in plaats van *C* beter *O* of *H* kiezen. Nemen we *H*, dan kunnen we weer een minteken aan het nummer laten voorafgaan om te voorkomen dat, in geval van 'kopiëren', ook aan het vaste punt een nieuw nummer wordt toegekend.

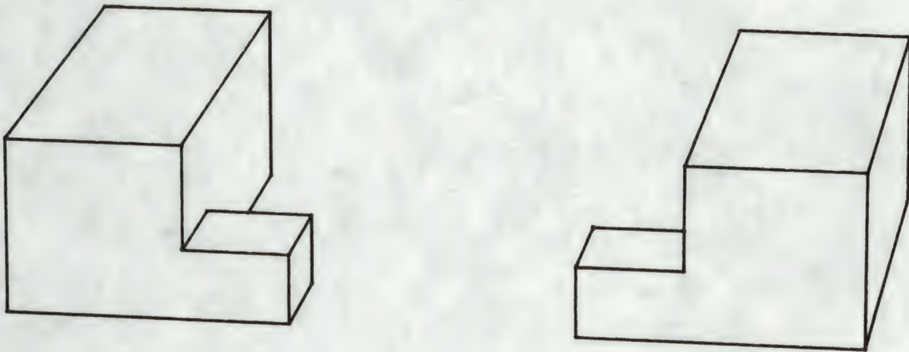


Figuur 1.22 Resultaat van schalen

Figuur 1.22 toont een voorbeeld van het schalen met een uniforme schaalfactor 2, waarbij wordt 'gekopieerd'. Door het vaste punt ver links van het oorspronkelijke (dat wil zeggen het linker) voorwerp te kiezen, verschijnt het nieuwe voorwerp, met een factor 2 vergroot, rechts van het oorspronkelijke.

1.8.4 Spiegeling

Een spiegeling is een transformatie die van een gegeven voorwerp een spiegelbeeld produceert. Een voorwerp wordt gespiegeld in een vlak, dat als een spiegel is te beschouwen. In figuur 1.23 is het rechter voorwerp verkregen door het linker te spiegelen in een verticaal vlak tussen de twee voorwerpen in.



Figuur 1.23 Resultaat van spiegeling

Met D3D is spiegeling gemakkelijk uit te voeren. Als, na commando *T*, ons antwoord op de vraag

```
Spiegeling? (J/N)
```

J is, dan verschijnen de volgende twee regels op het scherm:

```
PQR is spiegelvlak.
Puntnrs. P, Q, R:
```

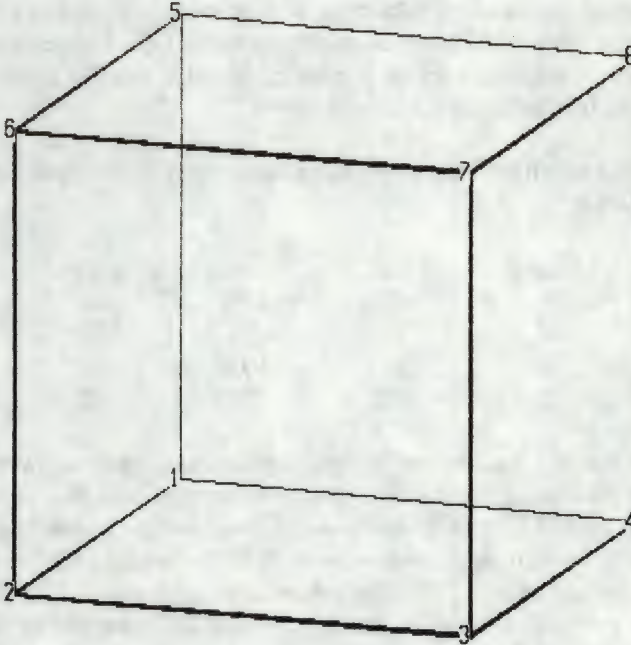
We geven dan de nummers van de punten P, Q en R op, die niet op dezelfde rechte lijn mogen liggen; het vlak door deze drie punten wordt gebruikt als vlak van spiegeling. Ook nu moeten we deze punten hebben gedefinieerd voordat we

commando *T* geven. Zij kunnen desgewenst punten van het te spiegelen voorwerp zijn. Net als de punten P en Q op de as van rotatie in paragraaf 1.8.1, zijn de punten P, Q en R in het vlak van spiegeling vaste punten. Ook bij een spiegeling kunnen we het voorwerp in kwestie 'verplaatsen' of 'kopiëren'. (Omdat een spiegelbeeld ontstaat moeten deze twee woorden hier niet te letterlijk worden opgevat.)

2 TOEPASSINGEN EN HULPPROGRAMMA'S

2.1 VOORWERPEN SAMENGESTELD UIT RECHTE PRISMA'S

De D3D-commando's *Read* en *Transform* (afgekort tot *R* en *T*) stellen ons in staat op een eenvoudige manier ingewikkelde voorwerpen op te bouwen, mits deze voorwerpen bestaan uit componenten waarvoor objectfiles beschikbaar zijn. Dank zij de mogelijkheid van niet-uniforme schaling kunnen we nog een stap verder gaan en één enkel basiselement vervormen tot elementen van verschillende vorm. Neem bijvoorbeeld de kubus met ribben van lengte 1, afgebeeld in figuur 2.1.



Figuur 2.1 Eenheidskubus

Nadat we deze kubus getekend hebben op de manier besproken in hoofdstuk 1, kunnen we commando *W* gebruiken om de file CUBE.DAT te produceren, die de volgende inhoud heeft:

```
1 0.000000 0.000000 0.000000
2 1.000000 0.000000 0.000000
3 1.000000 1.000000 0.000000
4 0.000000 1.000000 0.000000
5 0.000000 0.000000 1.000000
6 1.000000 0.000000 1.000000
7 1.000000 1.000000 1.000000
8 0.000000 1.000000 1.000000
Faces:
 2 3 7 6.
 4 1 5 8.
 3 4 8 7.
 1 2 6 5.
 6 7 8 5.
 3 2 1 4.
```

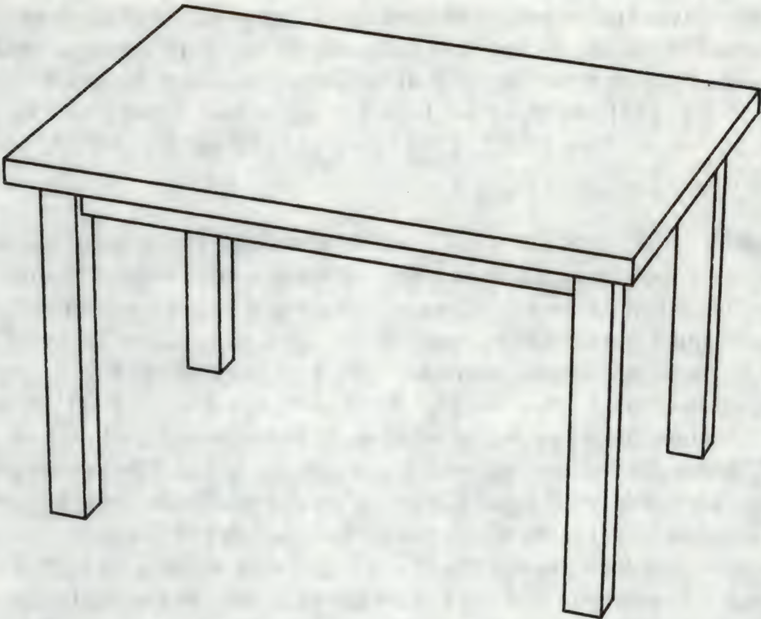
Dit lijkt misschien geen erg praktische bezigheid omdat kubussen in het dagelijks leven niet zoveel voorkomen. Maar we komen wel vaak allerlei prisma's (met rechthoekige grensvlakken) tegen en met programma D3D kunnen we deze, door van niet-uniforme schaling gebruik te maken, afleiden van een kubus. Laten we bijvoorbeeld de tafel in figuur 2.2 eens bekijken.

Deze bestaat uit negen rechte prisma's met afmetingen (in mm) zoals aangegeven in de volgende tabel:

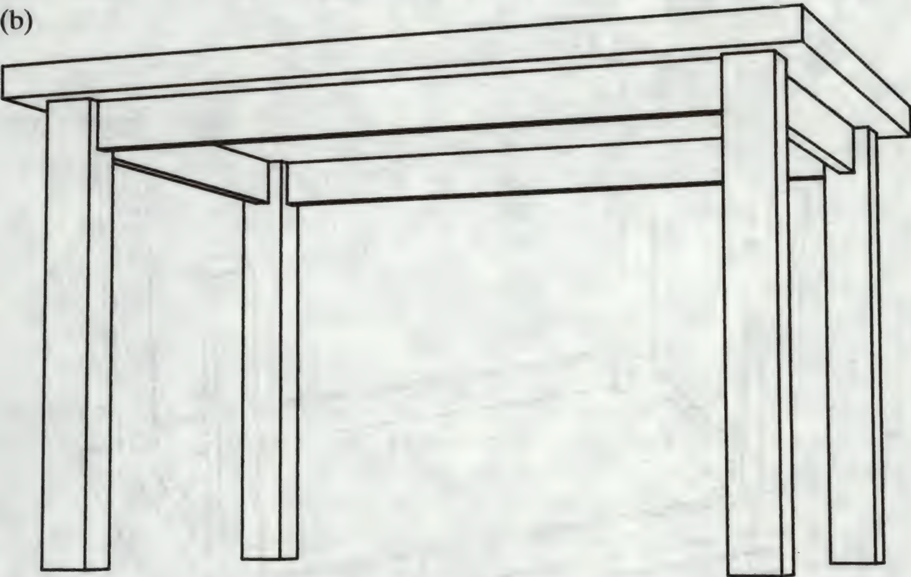
Onderdeel	Aantal	x-afm.	y-afm.	z-afm.
1	1	875	1250	50
2	4	75	75	725
3	2	25	1000	75
4	2	625	25	75

In plaats van 'lengte', 'breedte' en 'hoogte' spreken we hier liever over x-afmeting, y-afmeting en z-afmeting, omdat deze termen beter aangeven hoe elk onderdeel in de tafel wordt gebruikt. In D3D-terminologie uitgedrukt, zij vertellen ons hoe we niet-uniforme schaling moeten toepassen op de kubus van figuur 2.1. We verkrijgen bijvoorbeeld onderdeel 1, het tafelblad, door de schaalfactoren $S_x = 875$, $S_y = 1250$ en $S_z = 50$ te gebruiken wanneer we, na commando *T* gegeven te hebben niet-uniforme schaling hebben geselecteerd. Zoals gebruikelijk zijn er verschillende wegen die tot hetzelfde doel leiden, maar laten we ons beperken tot één manier om de (denkbeeldige) tafel te construeren. We zullen de oorsprong O van het

(a)



(b)



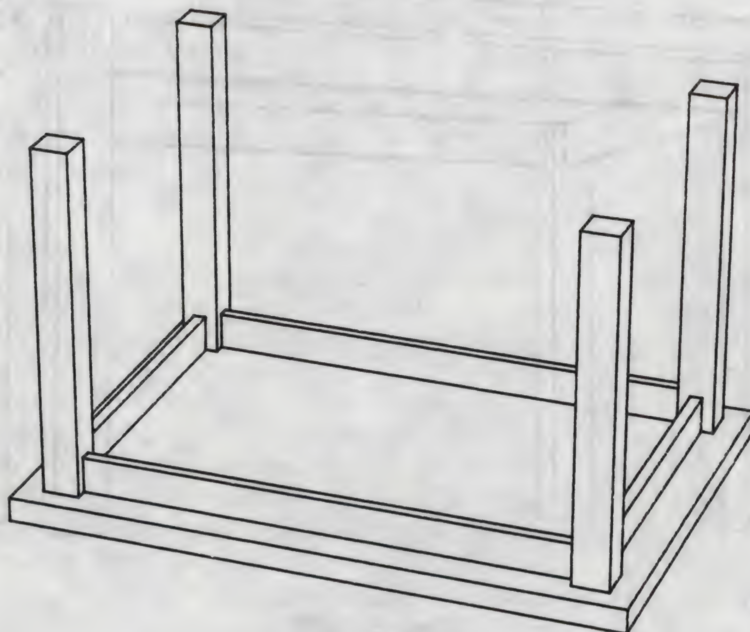
Figuur 2.2 Tafel

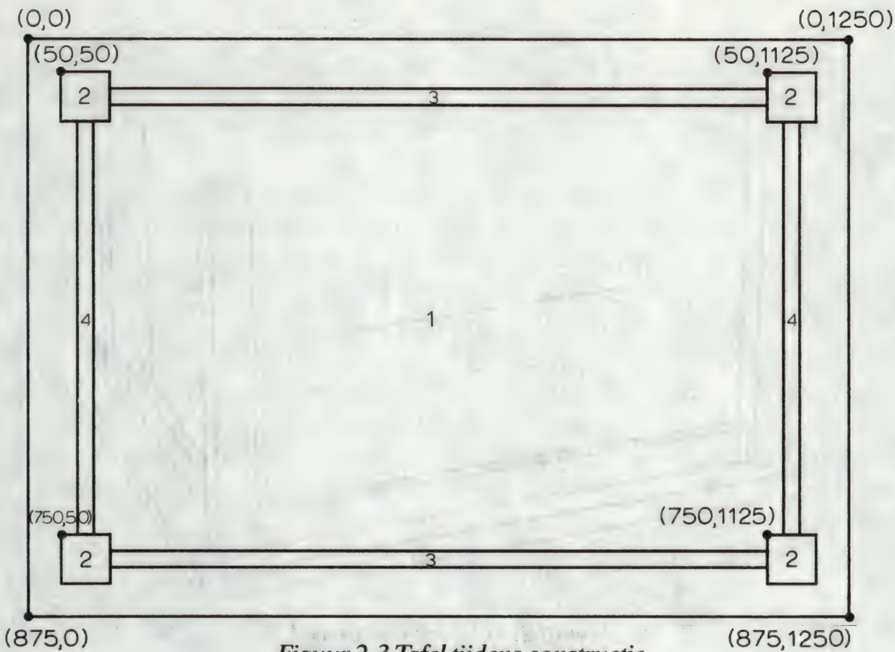
coördinatenstelsel gebruiken als het vaste punt van de schalingstransformaties die de eenheidskubus in de gewenste tafelonderdelen transformeren. Nadat we onderdeel 1 op deze manier door schaling hebben verkregen, bergen we het op in de file PART1.DAT, zodat we het later kunnen 'laden'. Op soortgelijke manier verkrijgen we de files PART2.DAT, PART3.DAT en PART4.DAT voor de onderdelen 2, 3 en 4.

We moeten nu de positie van het assenstelsel relatief ten opzichte van de tafel kiezen, maar voordat we dat doen zullen we enige aandacht aan de positie van de tafel zelf besteden. Als we een echte tafel van dit type moesten maken, dan zouden we waarschijnlijk het tafelblad ondersteboven op de vloer leggen en vervolgens de poten er bovenop zetten, enzovoort. In elke technische tekening moeten ingewikkelde details zo goed mogelijk zichtbaar zijn, wat in ons geval betekent dat figuur 2.2(a) niet voor ons doel geschikt is. In perspectivische tekeningen zijn we eraan gewend dat het oogpunt wat hoger gelegen is dan het voorwerp dat we bekijken, wat betekent dat figuur 2.2(b) ook minder geschikt is. Gelukkig stelt D3D ons in staat een omgekeerde tafel in zijn normale positie te brengen; dit kan òf door middel van een rotatie (over 180°, om een horizontale as), òf door middel van een spiegeling (in een horizontaal vlak), dus waarom zouden we niet beginnen met een omgekeerde tafel? Bekijk daarom de tafel in figuur 2.3(a).

Ons baserend op deze omgekeerde positie kunnen we figuur 2.3(b) een *bovenaanzicht* noemen. In technische tekeningen wordt vaak van dit soort

(a)



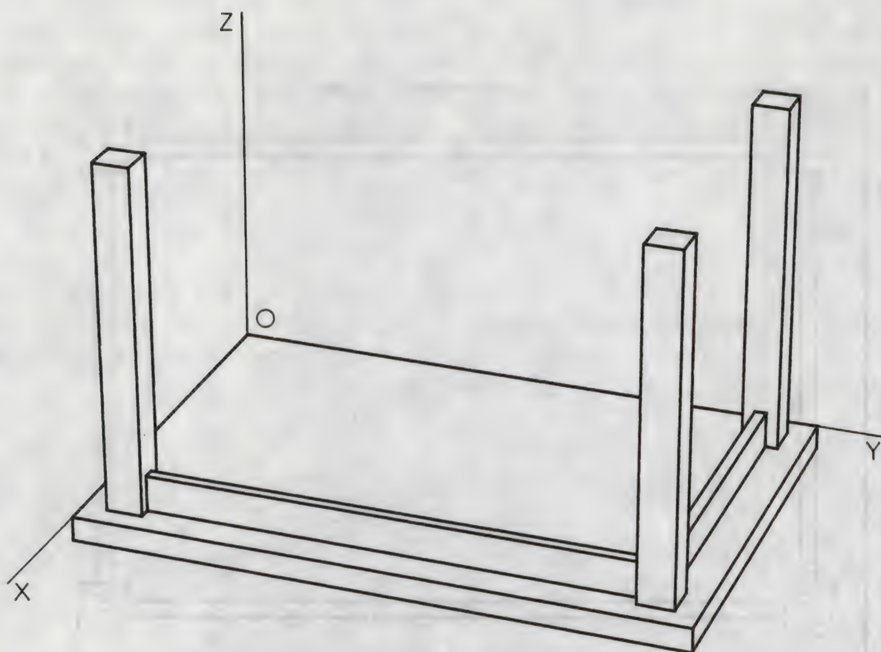


Figuur 2.3 Tafel tijdens constructie

aanzichten gebruik gemaakt, omdat zij veel gemakkelijker dan een correcte perspectivische afbeelding te vervaardigen zijn en omdat er veel lijnen in voorkomen die evenwijdig aan het vlak van tekening zijn en op ware lengte worden afgebeeld. Let erop dat figuur 2.3(b) ook de gebruikte onderdeelnummers laat zien.

Nu de tafel zich in een voor ons werk gerieflijke stand bevindt, kunnen we de positie van het assenstelsel kiezen. Laten we het scheidingsvlak tussen de poten en het tafelblad als het xy-vlak kiezen, zoals aangegeven in figuur 2.4. De posities van enkele belangrijke punten (met z-coördinaat gelijk aan 0), aangegeven in figuur 2.3(b), zijn gebaseerd op dit coördinatenstelsel. We moeten vooraf dit bovenaanzicht schetsen om te voorkomen dat we tijdens het echte constructieproces in de war raken.

Na al deze voorbereidingen kunnen we de verschillende onderdelen, opgeborgen in de files PART1.DAT, ..., PART4.DAT, gaan combineren en aldus de tafel opbouwen. We laden achtereenvolgens de onderdelen 1, 2, 3 en 4 (gebruik makend van commando *R*) en telkens verschuiven we het geladen onderdeel naar zijn juiste positie (met behulp van commando *T*). Iedere keer passen we een translatie toe op een zorgvuldig gekozen deelverzameling punten die we hebben verkregen hetzij door ze van een file te lezen, hetzij als resultaat van een vorige transformatie. We



Figuur 2.4 Coördinatenstelsel

beginnen met het laden van de file PART1.DAT (met commando *R*) en voeren een translatie uit met *delta* $x =$ *delta* $y = 0$, *delta* $z = -50$. Vervolgens laden we de file PART2.DAT: bij commando *R* beantwoorden we de vraag

```
Vervangen? (J/N):
```

met *N*, zodat de beide onderdelen 1 en 2 tegelijk op het scherm verschijnen. De hoekpunten van het zojuist geladen onderdeel 2 worden automatisch omgenummerd en het bereik van de nieuwe nummers (9-16) wordt getoond. Dit stelt ons in staat onderdeel 2 door middel van een translatie naar zijn juiste positie te verplaatsen (terwijl onderdeel 1 op zijn plaats blijft), waarbij we gebruik maken van

delta $x = 50$, *delta* $y = 50$, *delta* $z = 0$.

Voor de tweede tafelpoot kunnen we eenvoudig de *Copy*-mogelijkheid gebruiken en daarbij een translatie met

delta $x = 0$, *delta* $y = 1075$, *delta* $z = 0$

toepassen op het puntbereik (9-16) van de eerste poot, enzovoort. Wanneer de tafel, in omgekeerde stand, voltooid is, dan kunnen we hem spiegelen in een horizontaal vlak, laten we zeggen het vlak door de punten 1, 2 en 3, om hem in de

normale stand te krijgen. Daarna kunnen we met commando *H* een afbeelding van de tafel als massief lichaam verkrijgen. Natuurlijk moet dit worden voorafgegaan door commando *V* om een geschikt oogpunt te kiezen. De figuren 2.2(a) en (b) zijn op deze manier vervaardigd, met de volgende bolcoördinaten voor het oogpunt:

(a) $\rho = 7500$, $\theta = 20°$, $\varphi = 65°$;
(b) $\rho = 3750$, $\theta = 20°$, $\varphi = 95°$.

Tot zover onze discussie over het in elkaar zetten van een tafel. Het is slechts een voorbeeld van een voorwerp samengesteld uit alleen rechte prisma's. Er zijn nog bijzonder veel andere voorwerpen die uit alleen rechte prisma's bestaan, dus het nut van D3D blijft niet beperkt tot het ontwerpen van tafels.

2.2 ENKELE ANDERE STANDAARDCOMPONENTEN

We hebben zojuist gezien dat een kubus bijzonder nuttig gebruikt kan worden om er andere rechte prisma's van af te leiden. In veel toepassingen hebben we naast kubussen ook andere standaardcomponenten nodig. Zolang deze niet al te ingewikkeld zijn, kunnen we ze met D3D creëren, maar dit is niet altijd de meest efficiënte manier, vooral niet als deze een groot aantal grensvlakken hebben die gekromde oppervlakken benaderen. Een goed voorbeeld hiervan is een (rechte) cilinder. We zouden kunnen overwegen altijd uit te gaan van een cilinder met een hoogte en een diameter van 1, maar er doet zich hier een probleem voor waarvan we bij een kubus geen last hebben. Zoals eerder besproken, mogen we als grensvlakken alleen veelhoeken gebruiken, dus we moeten het (ronde) grondvlak van de cilinder benaderen door een regelmatige veelhoek met, laten we zeggen, *n* hoekpunten. De vraag doet zich dan voor, hoe groot *n* moet zijn. Hoe groter we *n* kiezen, hoe beter de benadering, maar hoe meer rekentijd en geheugenruimte het lichaam zal vergen. Het zou mooi zijn als we een willekeurige waarde van *n* zouden kunnen kiezen. Omdat programma D3D al complex genoeg is, is de mogelijkheid om een willekeurige benadering van een cilinder (en soortgelijke lichamen) te genereren niet opgenomen in dit programma zelf, maar zullen we in plaats hiervan een apart programma voor dit doel gebruiken. Zoals u misschien al gezien hebt in de directory van de bij dit boek verkrijgbare diskettes, staan hierop naast D3D verscheidene andere programma's. Eén hiervan is CYLINDER. Als u (nog) niet in het bezit bent van deze diskettes, dan kunt u het programma ook vinden in paragraaf 3.2. We starten dit programma door zijn naam in te typen:

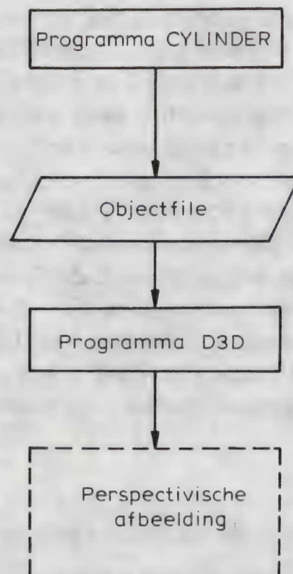
```
CYLINDER
```

Achtereenvolgens verschijnen nu de volgende vragen op het scherm:

```
Richting van de cilinderas? (X/Y/Z)
```

```
Diameter?
Hoogte?
Aantal hoekpunten in regelmatige veelhoek?
Naam van de uitvoerfile?
```

Als de cilinderas evenwijdig met een van de drie assen van het coördinatenstelsel moet zijn, dan typen we als antwoord op de eerste van de bovenstaande vijf vragen de naam (X, Y, of Z) van die as in. Na ook de overige vier vragen beantwoord te hebben ontstaat een file met alle gegevens van een cilinder die meteen in de goede stand staat, zodat geen rotatie nodig is. In de regel zal wel een translatie moeten worden uitgevoerd, omdat het centrum van de gegenereerde cilinder samenvalt met de oorsprong van het coördinatenstelsel, wat in de meeste gevallen niet gewenst is. Het antwoord op de vierde vraag bepaalt in feite de aard van het te genereren lichaam. Laten we het in te typen aantal hoekpunten n noemen. In werkelijkheid is het lichaam dan niet een cilinder, maar een regelmatig prisma met n zijvlakken; kiezen we bijvoorbeeld $n = 4$, dan krijgen we een prisma waarvan het grondvlak een vierkant is. Normaal zullen we een grotere waarde voor n nemen, zoals bijvoorbeeld 30. Het resultaat wordt geschreven in een *objectfile*, waarvan we de naam hebben ingetypt als antwoord op de laatste van de gestelde vragen. Deze file heeft het formaat dat door programma D3D wordt vereist en dat we hebben besproken in paragraaf 1.6. Dus nadat programma CYLINDER zijn werk gedaan heeft kunnen we D3D laten uitvoeren met de zojuist geproduceerde file als invoer, zoals figuur 2.5 illustreert.



Figuur 2.5 Objectfile, geproduceerd door CYLINDER en gebruikt door D3D

Als we bij het uitvoeren van programma CYLINDER de bovenstaande vijf vragen beantwoorden met

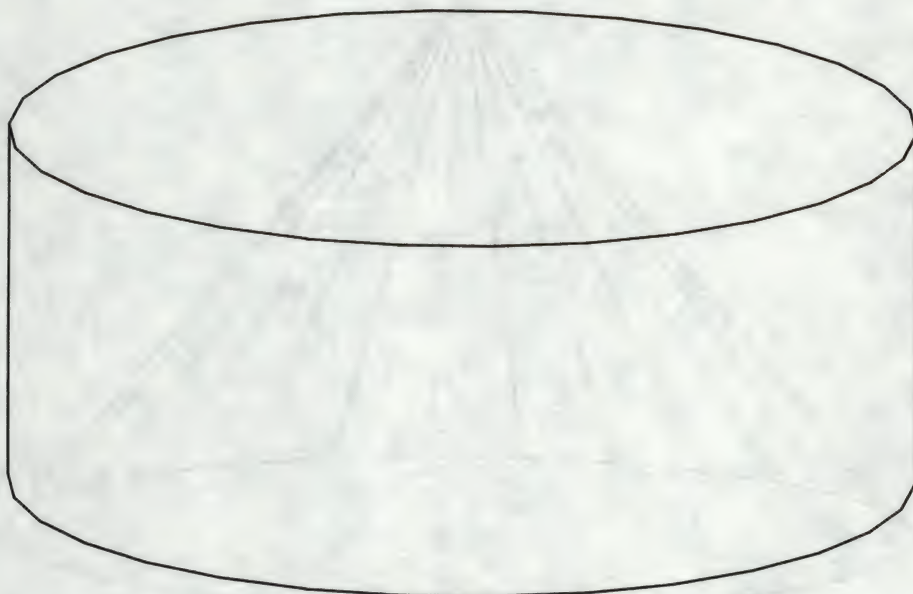
```
Z
5
2
3Ø
CYL.DAT
```

en daarna als volgt programma D3D starten

```
D3D CYL.DAT
```

dan zal commando *H* (met de drempelwaarde van 35°) figuur 2.6 opleveren. Hoewel deze afbeelding dat niet laat zien, valt de oorsprong van het coördinatenstelsel samen met het centrum (dat wil zeggen het zwaartepunt) van de cilinder.

Analoog aan het benaderen van een cilinder door een prisma, kunnen we een *kegel* benaderen door middel van een piramide. We zullen hiervoor programma CONE gebruiken; de tekst van dit programma is opgenomen in paragraaf 3.3. Dit programma genereert een (rechte) kegel, met zijn grondvlak in het xy-vlak en zijn top op de positieve z-as. Als we het programma starten, door



Figuur 2.6 Cilinder

```
CONE
```

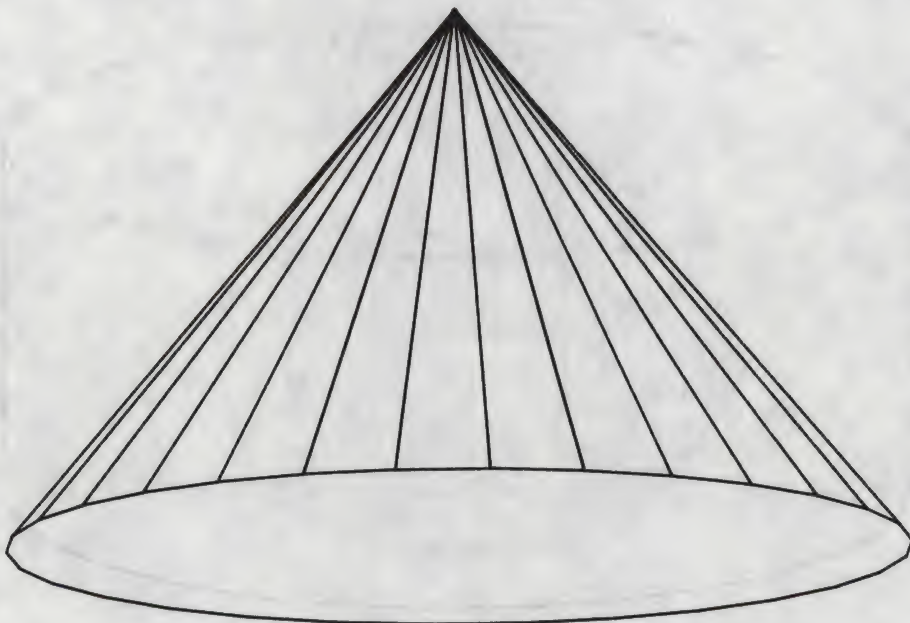
in te typen, dan verschijnen de volgende vragen (tot en met het vraagteken) op het scherm:

```
Diameter van grondvlak? 5
Hoogte? 3
Aantal hoekpunten van veelhoek? 30
Naam van de uitvoerfile? CONE.DAT
```

Als we deze vragen beantwoorden zoals hierboven aangegeven, dan zal daarna programma D3D, met de file CONE.DAT als invoer (en bij gebruik van de commando's *H* en een drempelwaarde van 0), een afbeelding van een kegel produceren zoals figuur 2.7 laat zien.

Let erop dat onze kegel een massief lichaam is. In figuur 2.7 bevindt het oogpunt zich iets lager dan de kegel, zodat we de kegel vanaf de onderkant bekijken. Daar het grondvlak ondoorzichtig is (en wel een regelmatige veelhoek met 30 hoekpunten) kunnen we alleen de schuine zijden van de driehoeken aan de voorkant zien, niet die van de achterkant.

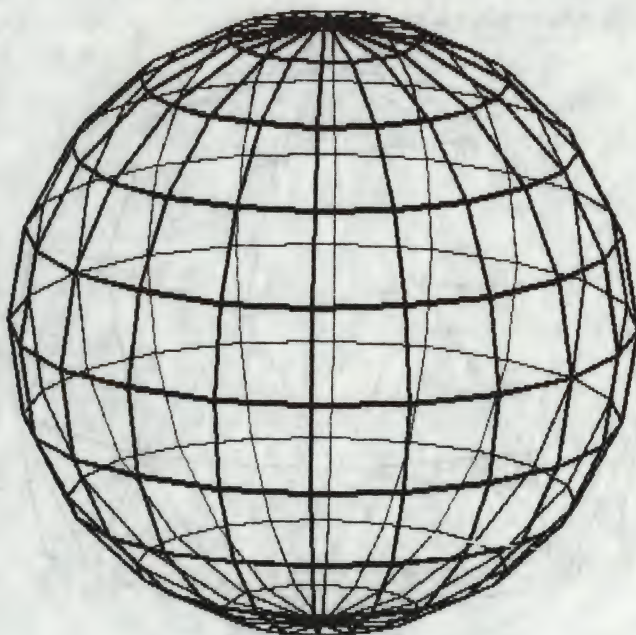
Een ander standaardelement is een *bol*. Er zijn verschillende manieren om een bol



Figuur 2.7 Een kegel

te benaderen met een lichaam begrensd door platte vlakken; wij zullen er twee bespreken die essentieel verschillend zijn. De eerste methode is gebaseerd op cirkels die we gewoonlijk op een globe aantreffen, zoals figuur 2.8 laat zien. We hebben twee polen, een noordpool bovenaan en een zuidpool onderaan; verder zijn er n verticale cirkels, de *meridianen*, die door de polen gaan. Al deze verticale cirkels hebben dezelfde straal r, die gelijk is aan de straal van de bol. De bol is verdeeld in m horizontale schijven, die iedere halve cirkel tussen de noord- en de zuidpool verdelen in m bogen van onderling gelijke lengte. Dit geeft m - 1 horizontale cirkels, de *breedtecirkels*. De twee polen en alle snijpunten van meridianen en breedtecirkels zijn de hoekpunten van veelhoeken die we zullen gebruiken als grensvlakken van het benaderende veelvlak. Deze veelhoeken zijn trapezia, behalve die waarvan de polen een hoekpunt zijn: de laatstbedoelde 'veelhoeken' zijn driehoeken. Hoe groter we n en m kiezen, hoe beter de benadering zal zijn. Maar als we n en m vergroten zal ook de rekentijd toenemen, vooral als we verborgen lijnen willen elimineren. De trapezia op de bol dichtbij de 'evenaar' (dat wil zeggen die welke deel uitmaken van de grootste schijf) zullen vierkanten benaderen als

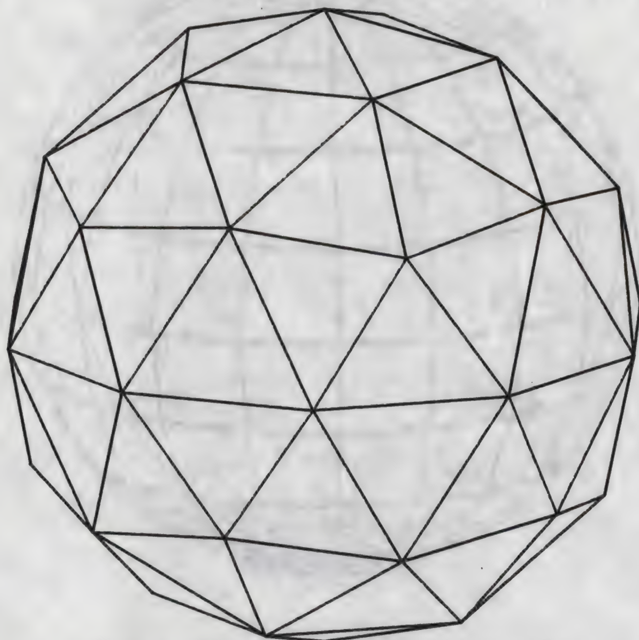
$$n = 2m,$$



Figuur 2.8 Benadering van een bol met $m = 10$ en $n = 20$

want we hebben n punten op elke horizontale cirkel en $2m$ punten op elke verticale cirkel. Programma SPHERE, waarvan de tekst in paragraaf 3.4 is opgenomen, genereert zo'n benadering van een bol. De gebruiker moet de waarden van m en n opgeven, evenals de straal van de bol en de naam van de uitvoerfile. Het middelpunt van de bol zal samenvallen met de oorsprong van het coördinatenstelsel en de polen liggen op de z-as. We verkrijgen de bol van figuur 2.8 door de programma's SPHERE (en D3D) te gebruiken met $m = 10$ en $n = 20$.

Hoewel dit plaatje er niet onaardig uitziet is het enigszins onbevredigend dat de grensvlakken niet ongeveer dezelfde grootte hebben. Om redenen van efficiency willen we tenslotte zo weinig mogelijk grensvlakken gebruiken (mits natuurlijk het ontstane lichaam niet al te veel van een echte bol gaat afwijken), dus het is niet efficiënt dat er zoveel kleine grensvlakken bij de polen zijn. Een ander mogelijk ongewenst effect is dat (in beelden waarin de meridianen en breedtecirkels zichtbaar zijn) de positie van het oogpunt het beeld sterk beïnvloedt. We zullen daarom een alternatieve methode om een bol te benaderen gaan bekijken, waarbij we een veelvlak met 80 driehoekige grensvlakken zullen gebruiken. In paragraaf 3.6 zullen we dit veelvlak meer in detail bespreken; hier zullen we het eenvoudigweg gebruiken. De file SPH80.DAT (gegenereerd door programma SPH80, zie paragraaf 3.6) is alles wat we nodig hebben. Het veelvlak in kwestie is afgebeeld in figuur 2.9. We kunnen daarna deze afbeelding verkrijgen door *D3D SPH80.DAT* in te typen en de commando's *H* en *P* te gebruiken.

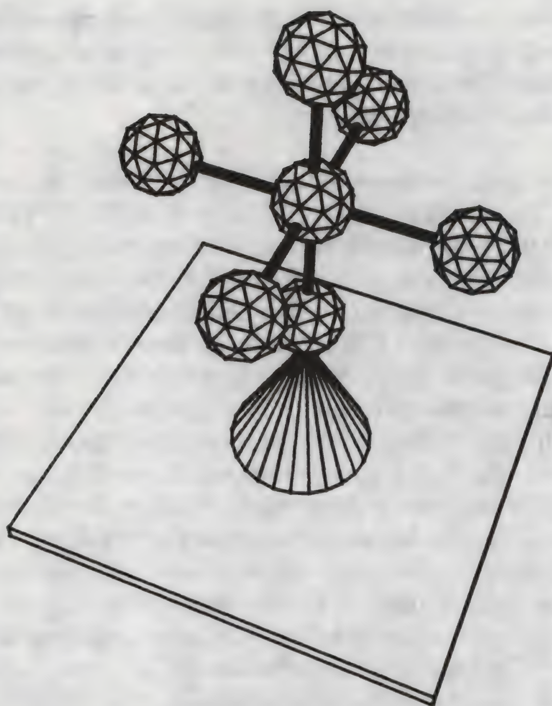


Figuur 2.9 Bol benaderd door 80 driehoeken

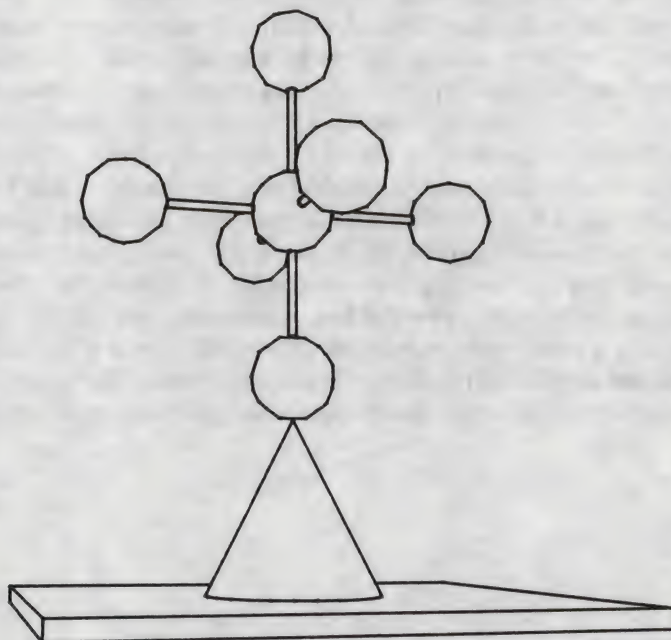
De figuren 2.10(a) en (b) tonen een combinatie van alle standaardelementen die we tot nu toe hebben besproken. Onderaan zien we een vierkante plaat waarop in het midden een kegel is geplaatst. Op deze kegel vinden we de onderste van zeven bollen, die verbonden zijn door zes dunne cilinders.

Elk van de zeven bollen wordt gerepresenteerd door een veelvlak met 80 driehoekige grensvlakken. De afbeelding is verkregen door de file SPH80.DAT maar één keer te lezen, namelijk voor de bol in het midden. Deze bol is 'gekopieerd' door middel van een translatie, waarbij de dichtstbijzijnde bol (met zijn middelpunt op de positieve x-as) werd verkregen. Toen is (één keer) een cilinder ingelezen vanaf een file (gegenereerd door programma CILINDER) en door middel van een translatie verplaatst naar zijn juiste positie tussen de twee bollen. Om enkele rotaties om de assen van het coördinatenstelsel te kunnen uitvoeren had ik vooraf de punten (1, 0, 0), (0, 1, 0) en (0, 0, 1) gedefinieerd en hieraan respectievelijk de nummers 1, 2 en 3 toegekend. (Zoals u zich zult herinneren, is punt 0 de oorsprong van het coördinatenstelsel.) Ik kon toen de dichtstbijzijnde bol en de cilinder over 180° om de y-as roteren om aldus de achterste bol en cilinder te verkrijgen. Omdat de laatste twee ontstane bollen en de twee cilinders een aaneengesloten rij hoekpuntnummers hebben, kon ik deze vier elementaire elementen (in één bewerking) over 90° om de z-as roteren om de bollen en cilinders op de positieve en negatieve y-as te verkrijgen. Tenslotte zorgde een rotatie van deze twee bollen en twee cilinders over 90° om de x-as voor de resterende bollen en cilinders op de z-as. In een complexe situatie zoals deze doen we er verstandig aan het voorwerp te splitsen in twee of meer delen en deze afzonderlijk in een file op te bergen. In dit voorbeeld heb ik twee van zulke files gebruikt, namelijk één voor de zeven bollen met hun verbindende cilinders en één voor de plaat en de kegel. Pas toen deze twee onderdelen klaar waren heb ik ze gecombineerd, door eerst het meest ingewikkelde onderdeel, de bollen en cilinders, en daarna het eenvoudiger onderdeel, de plaat en de kegel, in te lezen en het tweede onderdeel met een translatie naar beneden te verplaatsen. Figuur 2.10(b) toont hetzelfde voorwerp als figuur 2.10(a), maar bekeken vanuit een ander oogpunt; bij de eliminatie van verborgen lijnen is een drempelwaarde van 35° opgegeven, waardoor snijlijnen die niet in de overeenkomstige gekromde oppervlakken voorkomen ook in de figuur zijn weggelaten. Het is een kwestie van smaak of we zulke lijnen willen zien of niet; aangezien we in figuur 2.10(a) veel meer lijnstukken hebben dan in figuur 2.10(b) kost de eliminatie van verborgen lijnen in het eerste geval veel meer tijd dan in het laatste.

(a)



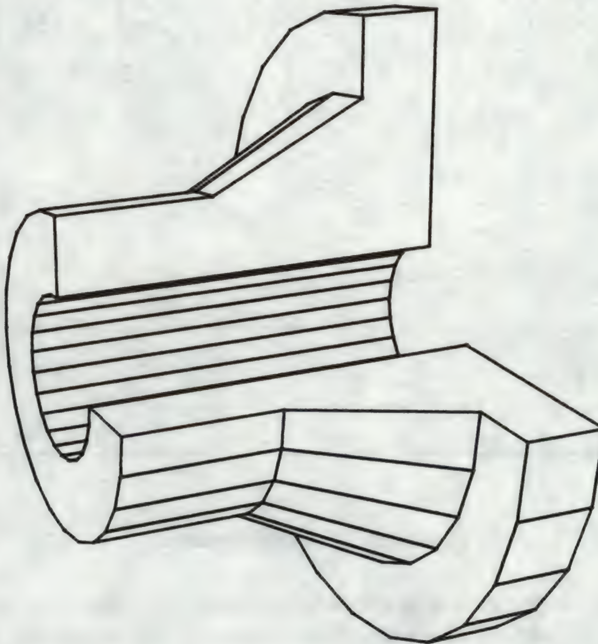
(b)



Figuur 2.10 Een voorwerp bestaande uit standaardelementen

2.3 OMWENTELINGSLICHAMEN EN 'CUTAWAY VIEWS'

In de vorige paragraaf hebben we enige programma's besproken voor het genereren van bijzondere *omwentelingslichamen*, namelijk een cilinder, een kegel en een bol. Deze programma's, geschreven in de programmeertaal C, zijn opgenomen in hoofdstuk 3. Als u onbekend bent met de taal C maar er wel in bent geïnteresseerd dan zou u een van de boeken over C kunnen bestuderen die in de literatuurlijst achterin zijn vermeld. Als u zover niet wilt gaan, dan zijn er nog twee andere mogelijkheden. Ten eerste kunt u, als u een andere programmeertaal beheerst, die gebruiken om invoerfiles voor D3D te genereren. Uiteindelijk gaat het alleen maar om de inhoud van die files en D3D heeft er geen weet van hoe deze zijn ontstaan. Ten tweede kan het natuurlijk ook zijn dat u zich in het geheel niet met het programmeren wilt of kunt bezighouden. Zelfs in dit geval kunt u D3D gebruiken voor het maken van interessante nieuwe voorwerpen, ook al kunnen deze niet worden samengesteld uit de beschikbare standaardelementen. Figuur 2.11 toont een voorbeeld van zo'n voorwerp. Het stelt een onvolledig omwentelingslichaam voor en we noemen de figuur een '*cutaway view*'. Het ontstaat door figuur 2.12 over een hoek van 270° te roteren om de as 1-2.

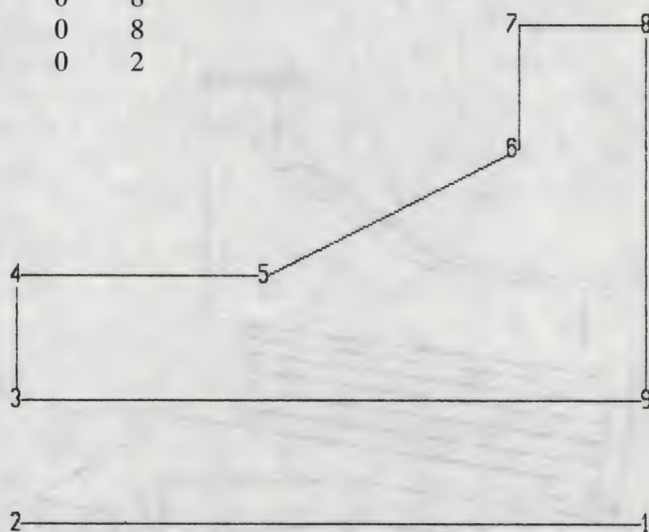


Figuur 2.11 'Cutaway view'

Het afgebeelde onvolledige voorwerp kan dienen om een blik te werpen in het inwendige van het overeenkomstige volledige voorwerp, dat we zouden verkregen hebben door de rotatie over 360° in plaats van over 270° uit te voeren.

Het maken van figuur 2.11, met gebruik van uitsluitend D3D, is niet moeilijk. We beginnen met het definiëren van de punten 1 t/m 9, aangegeven in figuur 2.12 (waarvoor het oogpunt met $\rho = 1000000$, $\theta = 90°$, $\varphi = 90°$ is gebruikt). De coördinaten van deze punten zijn als volgt:

	x	y	z
1	0	0	0
2	10	0	0
3	10	0	2
4	10	0	4
5	6	0	4
6	2	0	6
7	2	0	8
8	0	0	8
9	0	0	2



Figuur 2.12 Langsdoorsnede

Dus punt 1 is de oorsprong van het coördinatenstelsel en punt 2 ligt op de positieve x-as (in figuur 2.12 wijzend naar links); alle y-coördinaten zijn nul, dus we werken in het xz-vlak. We willen nog geen lijnen zien, dus in werkelijkheid beginnen we met een versie van figuur 2.12 met alleen punten en geen lijnen. Gebruik makend van commando *T* roteren we alle punten behalve 1 en 2 over een hoek van 15° om de

x-as. Wanneer daarom wordt gevraagd, geven we de nummers 3 en 9 op als onder- en bovengrens en, na opgegeven te hebben dat we willen roteren, geven we de punten 1 en 2 op als de punten P en Q die vereist zijn om de as van rotatie vast te leggen. In deze en alle volgende rotaties gebruiken we 'Copy' (niet 'Move'); we zullen dit niet steeds opnieuw vermelden, evenmin als het feit dat we steeds de punten 1 en 2 gebruiken om de rotatie-as op te geven. Het nummer van elk nieuw punt wordt verkregen door het nummer van het overeenkomstige oude punt met 7 te verhogen, hetgeen de puntnummers 10 t/m 16 oplevert. Dit stelt ons in staat als volgt zeven grensvlakken te definiëren:

```
F
3  -4  11  -10.
4   5  12   11.
5   6  13   12.
6  -7  14  -13.
7   8  15   14.
8  -9  16  -15.
9   3  10   16.
```

Het minteken voorkomt dat een lijnstuk in de figuur wordt getekend. Zo behoort bijvoorbeeld lijnstuk (11, 10) niet zichtbaar te zijn in het eindresultaat, figuur 2.11. Hoewel lijnstuk (3, 4) uiteindelijk wel getekend moet worden, wordt het hier op dezelfde manier behandeld, wat te maken heeft met nog uit te voeren kopieerbewerkingen. Afgezien van enkele ontbrekende lijnstukken hebben we nu een gedeelte van het voorwerp gecreëerd dat we een *sector* van 15° zullen noemen. Deze sector moeten we nu laten aangroeien tot een sector van 270°; hoewel we dit zouden kunnen doen door de sector van 15° zeventien (= 270/15 − 1) keer kopiërend te roteren, zullen we ons eindresultaat in slechts vijf rotaties verkrijgen, zoals tabel 2.1 laat zien.

Tabel 2.1 Rotaties om een sector te vergroten van 15° tot 270°.

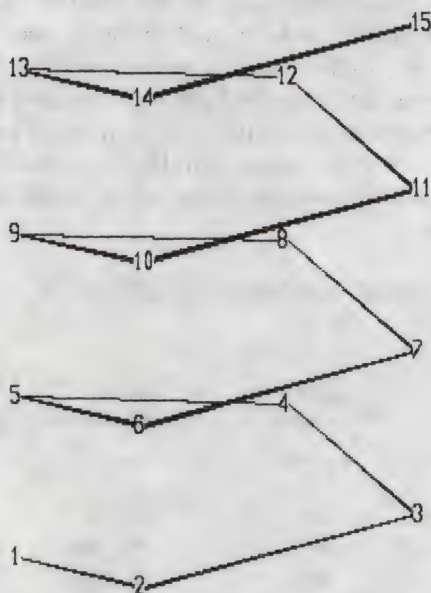
Ondergrens	Bovengrens	Hoek van rotatie	Nieuw puntbereik	Resulterende sector
3	16	15°	17- 30	30°
3	30	30°	31- 58	60°
31	58	30°	59- 86	90°
3	86	90°	87-170	180°
87	170	90°	171-254	270°

In elke stap roteren (en kopiëren) we een sector met een hoek gelijk aan de hoek van rotatie, genoemd in de derde kolom. Na al deze rotaties moeten we een horizontaal en een verticaal grensvlak toevoegen:

```
F
9  8  7  6  5  4  3.
248 249 250 251 252 253 254.
```

Zoals u zich herinnert moeten we bij het opgeven van de hoekpuntnummers van een grensvlak, als we vanaf de buitenkant naar het voorwerp kijken, de hoekpunten anti-kloksgewijs doorlopen, vandaar de dalende volgorde van de eerste en de stijgende volgorde van de tweede rij hoekpuntnummers.

Pas na het geven van commando *H*, dat de verborgen lijnen verwijdert, krijgen we een beeld dat er toonbaar uitziet, zie figuur 2.11. De bolcoördinaten van het gebruikte oogpunt zijn $\rho=50$, $\theta=65°$, $\varphi=70°$. Gedurende het constructieproces wordt het voorwerp getoond als draadmodel, waarbij ook nog enkele grensvlakken ontbreken. Omdat er zoveel punten zijn kunnen we beter geen hoekpuntnummers of assen in de beeldfiguur opnemen, dus vooraf zal commando *M* gebruikt dienen te worden.



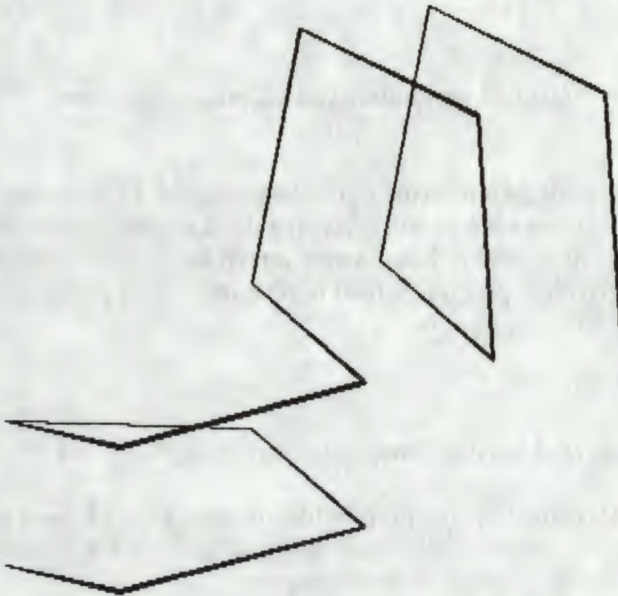
Figuur 2.13 Enkele punten op een veer

2.4 VLOEIENDE DRIEDIMENSIONALE KROMMEN

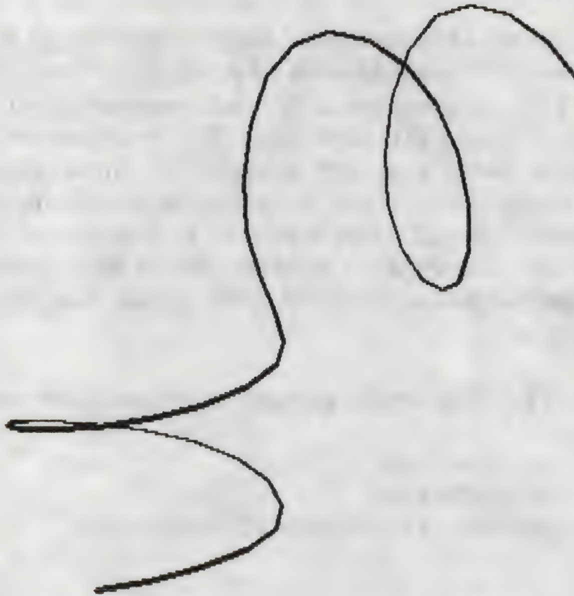
Er is een wiskundige benaderingsmethode, B-spline approximatie, die gebruikt kan worden om zonder veel moeite vloeiende (ofwel 'gladde') krommen te verkrijgen. We zullen deze methode gebruiken in de driedimensionale ruimte, hetgeen een kromme zoals die in figuur 2.15 kan opleveren. We hebben maar een beperkt aantal punten nodig, die we met D3D kunnen definiëren. Met commando *W* schrijven we de coördinaten van die punten in een file en deze file wordt dan gelezen door een nieuw programma, CURVE3, waarvan de tekst in paragraaf 3.8 gegeven is. Dit programma genereert een groot aantal andere punten door middel van B-spline approximatie; deze techniek wordt hier slechts besproken voor zover de gebruiker ermee te maken heeft.

Programma CURVE3 vraagt om drie gegevens die we moeten intypen:

1 De naam van een invoerfile
2 De naam van een uitvoerfile
3 Een geheel getal *N*, dat we hieronder zullen bespreken.



Figuur 2.14 Resultaat na het roteren van de punten 9 t/m 15



Figuur 2.15 Resultaat van B-spline-approximatie

Zowel de invoer- als de uitvoerfile zijn wat we noemen 'D3D-compatibel', dat wil zeggen zij hebben het formaat van objectfiles die door D3D gelezen en geschreven worden. Het gehele getal N is het aantal intervallen tussen twee opeenvolgende gegeven punten: bij m gegeven punten in de invoerfile zal de uitvoerfile k punten bevatten, waarbij

$$k = (m - 3)N + 1$$

zoals we aan het eind van deze paragraaf zullen zien.

Dank zij de D3D-compatibiliteit van de files die door CURVE3 worden gelezen en geschreven kunnen we met D3D zowel de invoerfile van CURVE3 klaarmaken als de geproduceerde kromme grafisch weergeven:

D3D → file → CURVE3 → file → D3D

Van de invoerfile wordt alleen het eerste gedeelte, waarin de nummers en de

coördinaten van alle punten staan, gebruikt door CURVE3. Een tweede gedeelte, beginnend met *Faces*:, kan wel of niet aanwezig zijn. Als het er is wordt het eenvoudig genegeerd. Programma CURVE3 negeert overigens ook de puntnummers, hetgeen betekent dat bijvoorbeeld de file-inhoud

```
9    1.5  3.5  2.5
5    1.0  3.0  2.0
7    0.5  0.7  2.5
8    0.1  0.2  0.0
```

voor CURVE3 equivalent is met

```
1    1.5  3.5  2.5
2    1.0  3.0  2.0
3    0.5  0.7  2.5
4    0.1  0.2  0.0
```

Gelukkig staan de punten in de uitvoerfile van D3D gerangschikt volgens opklimmend puntnummer, zodat D3D nooit de (mogelijk verwarrende) eerstgenoemde file-inhoud zal afleveren. U zou evenwel zo'n file met andere middelen, zoals een editor of een zelfgemaakt programma, kunnen genereren en dan moet u weten dat de regels in de file in de goede volgorde moeten staan. Let erop dat deze eis niet gesteld wordt door D3D: dat programma let wel op de puntnummers in de invoerfile, zodat daar de regels in het deel vóór '*Faces*' in willekeurige volgorde mogen staan.

De figuren 2.13, 2.14 en 2.15 zijn in die volgorde verkregen. Figuur 2.13 toont de punten 1 t/m 15, die op een spiraalveer liggen, en de lijnstukken die ze met elkaar verbinden. Zoals u zich zult herinneren worden zulke lijnstukken in objectfiles gerepresenteerd door getallenparen na het woord *Faces*, of, wanneer we D3D interactief aan het gebruiken zijn, na commando *F*, zoals bijvoorbeeld in

```
F
1 2.
2 3.
3 4.
```

Zoals zojuist is vermeld, worden deze lijnstukken genegeerd door programma CURVE3; ik heb ze toch in de file opgenomen om figuur 2.13 zo duidelijk mogelijk te maken. Daarna heb ik een transformatie (met 'move') gebruikt en wel een rotatie van de puntenrij met ondergrens 9 en bovengrens 15, over een hoek van 90° en om de lijn door de punten 7 en 8. Het resulterende voorwerp, getoond in figuur 2.14 werd geschreven in een file, laten we zeggen TEST.DAT. Daarna heb ik programma CURVE3 op deze file toegepast. Na het starten van het programma, komen altijd

drie vragen op het scherm, die, tezamen met de antwoorden die bij ons voorbeeld behoren, hieronder zijn weergegeven:

```
Invoerfile:  TEST.DAT
Uitvoerfile: TEST1.DAT
Geef N, het aantal intervallen tussen
twee opeenvolgende gegeven punten: 4
```

Door daarna te typen

```
D3D TEST1.DAT
```

en de commando's *E* en *P* in te typen werd figuur 2.15 verkregen. In plaats van slechts 15 punten in de file *TEST.DAT* zijn er 49 punten in de file *TEST1.DAT.* Al deze punten zijn berekend door middel van B-spline approximatie; hoewel u de wiskundige details hiervan niet hoeft te kennen zijn er twee punten die u moet onthouden. U zult ze misschien niet leuk vinden maar zij vormen de prijs die we moeten betalen voor de buitengewone 'gladheid' van de kromme:

1 In het algemeen gaat de kromme niet precies door de gegeven punten: hij benadert ze slechts. Hoe dichter de gegeven punten bij elkaar liggen, hoe beter de benadering zal zijn, dus als de kromme de punten niet goed genoeg naar onze zin benadert dan kunnen we daar verbetering in brengen door meer punten op te geven.

2 Het allereerste en het allerlaatste punt worden niet benaderd. In ons voorbeeld benadert het eind van de kromme onderaan in figuur 2.15 punt 2, zie figuur 2.13; het andere eind, bovenaan, benadert het op een na laatste punt (punt 14) van de gebroken lijn in figuur 2.14. Het zal duidelijk zijn dat we ten minste vier punten, zeg A, B, C en D, nodig hebben om de B-spline methode te kunnen toepassen. In dàt geval ligt de resulterende kromme ongeveer tussen B en C. Hoewel de punten A en D niet worden benaderd, leveren zij wel een bijdrage in de berekening, dus we kunnen ze niet missen.

Zonder geavanceerde wiskunde te gebruiken kunnen we nu bovengenoemde formule

$$k = (m - 3)N + 1$$

begrijpen. Er zijn *m* gegeven punten, waarvan het eerste en het laatste niet worden benaderd, dus slechts $m - 2$ ervan zullen bij benadering door de kromme met elkaar worden verbonden. We stellen ons eerst voor dat zij door middel van rechte

lijnstukken inderdaad verbonden zijn; aangezien het aantal verbindingslijnstukken altijd 1 kleiner is dan het aantal verbonden punten, zijn er $m - 3$ van zulke lijnstukken. Elk ervan wordt verdeeld in N intervallen, hetgeen tezamen $(m - 3)N$ kleine lijnstukjes geeft. Deze verbinden de k nieuwe punten waarom het gaat; als we nu weer bedenken dat het aantal verbindingen 1 kleiner is dan het aantal verbonden punten, dan vinden we bovenstaande uitdrukking voor k.
Let erop dat wat we in deze paragraaf een *kromme* hebben genoemd in werkelijkheid bestaat uit een aantal rechte lijnstukken. Door N groot genoeg te kiezen, zullen deze lijnstukken heel klein zijn, zodat we, zoals figuur 2.15 laat zien, een heel goede benadering van een echte kromme kunnen verkrijgen.

2.5 KABELS EN KNOPEN

De driedimensionale krommen van paragraaf 2.4 zijn wiskundige abstracties: we kunnen ze beschouwen als draden met diameter 0. We willen er nu echte massieve lichamen van maken, dat wil zeggen draden of *kabels*, met een diameter groter dan 0. Op dezelfde manier als we een cilinder kunnen opvatten als een rechte lijn (de cilinderas) waaraan we een zekere dikte hebben gegeven, zullen we kabels vormen uit driedimensionale krommen. De analogie tussen onze cilinders en onze kabels is ook van toepassing op de benaderingsmethode die we zullen toepassen: zoals gebruikelijk, zullen we een gekromd oppervlak benaderen door een groot aantal platte grensvlakken.
Net als in de vorige paragraaf zullen we een programma gebruiken met een in- en een uitvoerfile, die beide D3D-compatibel zijn. Dit programma, CABLE geheten, is opgenomen in paragraaf 3.9. De invoerfile ervoor kunnen we het best genereren met programma CURVE3, zie paragraaf 2.4; daar de uitvoerfile normaal gesproken door D3D zal worden gelezen zullen we dus gewoonlijk achtereenvolgens programma D3D, CURVE3, CABLE en weer D3D gebruiken. (We kunnen natuurlijk de kromme eerst op het scherm bekijken voordat we CABLE erop toepassen, wat betekent dat we dan D3D ook nog gebruiken tussen CURVE3 en CABLE in). Net als CURVE3 stelt CABLE ons enkele vragen, namelijk de naam van de in- en uitvoerfile, het aantal punten die op elke cirkel gekozen moeten worden en de diameter van zo'n cirkel. Deze cirkels zijn analoog aan het grondvlak van een cilinder of een kegel: als we op iedere cirkel n punten hebben dan is elke dwarsdoorsnede van de kabel een regelmatige veelhoek met n hoekpunten. Als we n groot kiezen zullen we een heel goede benadering van een kabel krijgen, maar we moeten niet vergeten dat we uiteindelijk alle verborgen lijnen willen elimineren, wat wellicht tijdrovend is als er erg veel grensvlakken zijn. Zoals we aan het eind van de vorige paragraaf hebben gezien, produceert CURVE3 niet minder dan $(m - 3)N$ kleine lijnstukken, waarvan nu elk zal expanderen tot n grensvlakken, zodat er

$$(m - 3)Nn$$

grensvlakken zullen zijn om het gekromde kabeloppervlak te benaderen. Voor de duidelijkheid recapituleren we:

m = het aantal voor de B-spline-approximatie gegeven punten, dat wil zeggen het aantal punten in de file die door programma CURVE3 wordt gelezen;

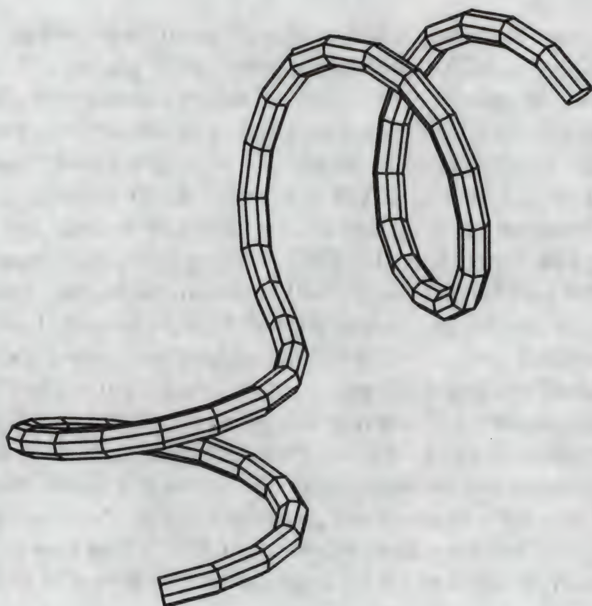
N = het aantal intervallen tussen twee opeenvolgende punten (N wordt ingetypt wanneer we programma CURVE3 uitvoeren);

n = het aantal punten op elke dwarsdoorsnede van de kabel (n wordt ingetypt als we programma CABLE uitvoeren);

Het is aan te bevelen met een vrij kleine waarde van n te beginnen en, als het resultaat onbevredigend is, daarna die waarde wat groter te kiezen. Om figuur 2.16 te verkrijgen heb ik $n=8$ gebruikt. Het aantal grensvlakken voor het benaderen van het kabeloppervlak was dus hier gelijk aan

$$(15-3) \times 4 \times 8 = 384.$$

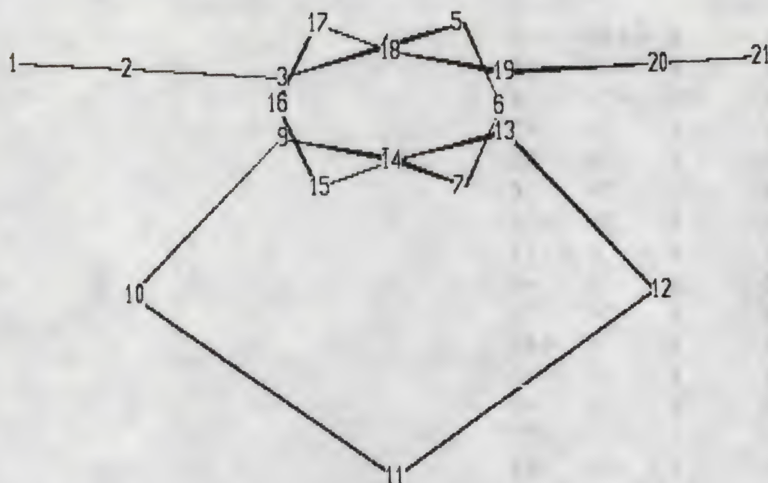
Hoewel het onderscheid tussen dikke en dunne lijnen ons in staat stelt enige 'diepte' te zien in de kromme van figuur 2.15, geeft figuur 2.16 de driedimensionale situatie veel duidelijker weer.



Figuur 2.16 Kabel

We kunnen de programma's CURVE3 en CABLE gebruiken om driedimensionale plaatjes van *knopen* te maken. Bekijk bijvoorbeeld figuur 2.17 eens, tezamen met de volgende erbij behorende file met invoergegevens voor programma CURVE3:

```
 1    0  -100    12
 2    0   -70    10
 3  -15   -30     8
 4   15     0    15
 5    0    20    22
 6  -15    30     0
 7    0    20   -22
 8   15     0   -15
 9  -15   -30    -8
10    0   -70   -50
11    0     0  -100
12    0    70   -50
13   15    30    -8
14  -15     0   -15
15    0   -20   -22
16   15   -30     0
17    0   -20    22
18  -15     0    15
19   15    30     8
20    0    70    10
21    0   100    12
Faces:
 1    2.
 2    3.
 3    4.
 4    5.
 5    6.
 6    7.
 7    8.
 8    9.
 9   10.
10   11.
11   12.
12   13.
13   14.
14   15.
15   16.
16   17.
17   18.
18   19.
19   20.
20   21.
```



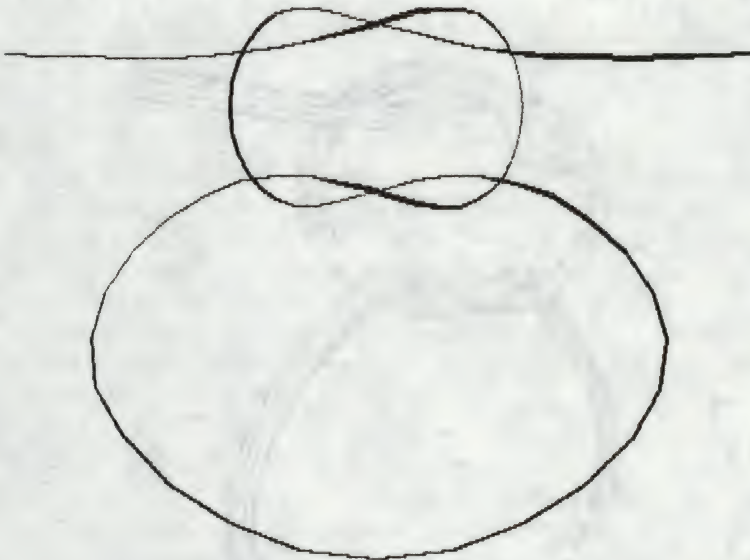
Figuur 2.17 Invoergegevens voor CURVE3 afgebeeld met D3D

We kunnen deze file zowel met D3D als met een gewone editor verkrijgen; ook kunnen we deze beide methoden combineren: eerst gebruiken we D3D om iets als figuur 2.17 te produceren (met $\theta = 0°$ en $\varphi = 90°$) maar waarbij alle punten nog in het yz-vlak liggen en daarna gebruiken we een editor om de x-coördinaten die een waarde ongelijk aan nul moeten krijgen te veranderen. Zo hebben bijvoorbeeld de punten 4 en 18 aanvankelijk dezelfde coördinaten, namelijk $x=0$, $y=0$, $z=15$. (Zoals u zich zult herinneren gebruiken we een rechts coördinatenstelsel, waarbij de positieve x-as naar ons toe wijst.) Het maken van deze file is misschien een beetje vervelend, maar het is bepaald geen monnikenwerk; als we dit eenmaal hebben gedaan, dan doet de computer het zware werk voor ons, namelijk het tekenen van een platte knoop, bekeken vanuit een willekeurig oogpunt. Nadat we programma CURVE3 op deze file hebben toegepast, met bijvoorbeeld $N=5$, verkrijgen we figuur 2.18, met een bijbehorende objectfile. Op deze laatste file passen we dan programma CABLE toe, hetgeen het eindprodukt, getoond in de figuren 2.19(a), (b) en (c), oplevert. Voor elke cirkel heb ik een straal $r=2$ genomen, met daarop $n=10$ punten. Het meeste werk wordt verricht programma D3D, en wel wanneer de verborgen lijnen verwijderd moeten worden. In gevallen als deze hangt het van de snelheid van uw computer af welke waarden van N en n u

redelijkerwijze kunt gebruiken. Het kan allemaal op een machine met slechts een 8088-processor, maar als u daarna overschakelt naar een IBM PC/AT met een mathematische co-processor, dan scheelt dat enorm. We kunnen de hoeveelheid rekentijd ook reduceren, zij het minder sterk dan op de zojuist genoemde wijze, door een vrij grote waarde te kiezen voor de 'drempelhoek' α, waardoor in het eindresultaat zekere kunstmatig geïntroduceerde randen worden weggelaten. Voor de figuren 2.19(a) en (b) heb ik $\alpha = 35°$ gebruikt en voor figuur 2.19(c) de waarde $\alpha = 50°$. Let erop dat door $\alpha = 35°$ te gebruiken de randen evenwijdig aan de as van de kabel getekend worden, omdat we werken met $n = 10$ en dus

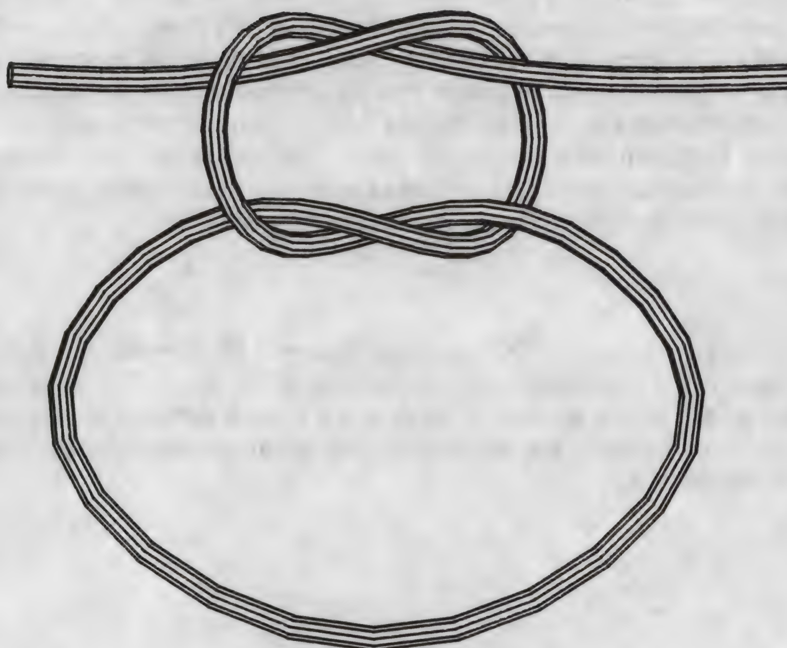
$$360°/n = 36° > 35°.$$

Door de vrij grote hoek $\alpha = 35°$ zijn in de figuren 2.19(a) en (b) de randen op dwarsdoorsneden weggelaten. Zij zouden, net als in figuur 2.16 in de figuur verschenen zijn als we de hoek α gelijk aan 0 hadden gekozen, of, equivalent hiermee, als we de vraag na commando *H* over gekromde oppervlakken negatief hadden beantwoord.

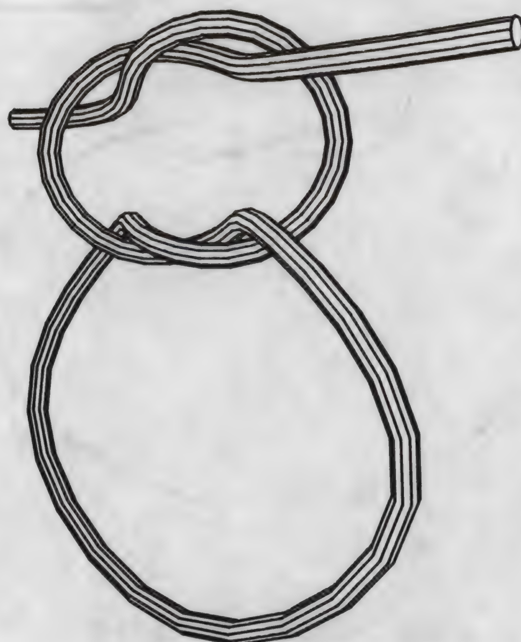


Figuur 2.18 Resultaat van CURVE3

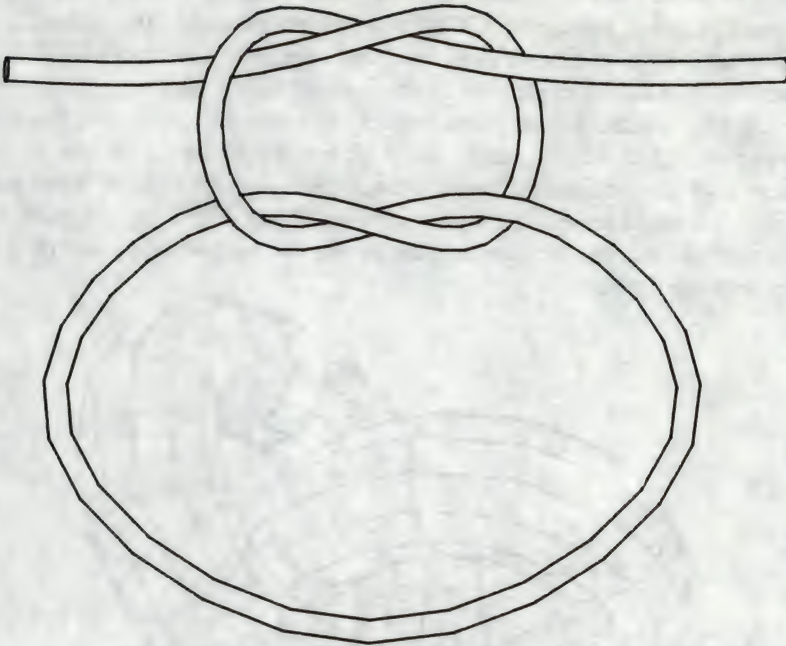
(a)



(b)



(c)

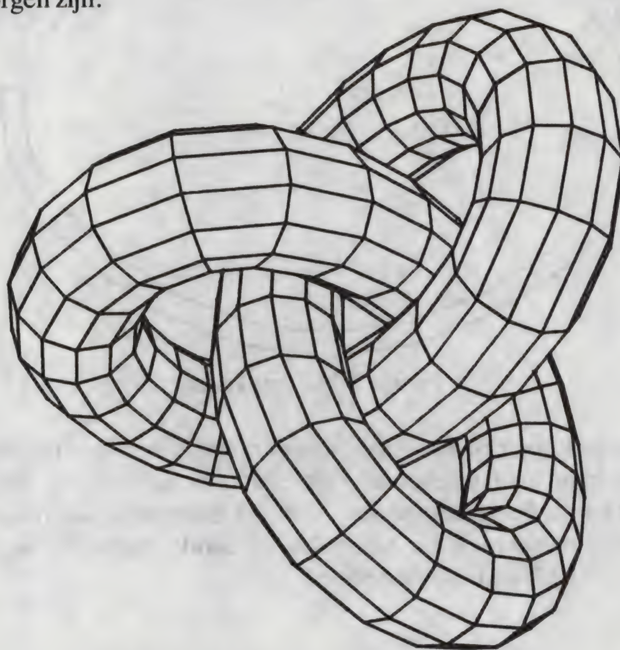


Figuur 2.19 Platte knoop

Figuur 2.20 toont nog een resultaat (vrijwel identiek met een tekening die M.C. Escher met de hand heeft getekend!). Dit resultaat is verkregen op dezelfde manier als de zojuist besproken platte knoop, namelijk door eerst programma CURVE3 uit te voeren, en wel met de hieronder getoonde invoerfile; daarna zijn de programma's CABLE en D3D uitgevoerd.

```
 1    0   1.73  -1
 2   -2   1.73   1
 3    2   0      2
 4    0  -1.73   1
 5   -2   0      0
 6    2   1.73   1
 7    0   1.73   3
 8   -2   0      2
 9    2   0      0
10    0   1.73  -1
11   -2   1.73   1
12    2   0      2
```

In tegenstelling tot figuur 2.19 heb ik hier $\alpha = 0°$ gebruikt, zodat in figuur 2.20 alle zichtbare randen getekend zijn. Omdat de programma's CURVE3 en CABLE eigenlijk bedoeld zijn voor eindige krommen en kabels, spreekt het niet vanzelf dat de oneindige 'kabel' van figuur 2.20 op deze manier gemaakt kan worden. Zoals we in paragraaf 3.8 zullen zien, is er in dit opzicht geen probleem met CURVE3. Als we evenwel (met CABLE) de kromme een zekere dikte gaan geven, dan gebruiken we regelmatige veelhoeken in plaats van cirkels; we mogen dan niet verwachten dat de hoekpunten van de allerlaatste veelhoek precies samenvallen met die van de allereerste. Ik moet daarom bekennen dat de twee uiteinden van de kabel in figuur 2.20 niet helemaal netjes in elkaar overgaan, maar dat ik het oogpunt zo heb gekozen dat die niet perfect samenvallende uiteinden achter een ander deel van de knoop verborgen zijn.



Figuur 2.20 Knoop naar M.C. Escher

2.6 VIERKANT SCHROEFDRAAD

We zullen ons in deze paragraaf bezighouden met een programma dat mogelijk enig nut zou kunnen hebben in de werktuigbouwkunde. Het gaat om schroefdraad met een vierkant profiel, zoals de figuren 2.21(a) en (b) laten zien. Het programma, SCRTHR geheten, produceert een objectfile die door D3D gelezen kan worden om er een driedimensionale weergave van te maken. Het programma is opgenomen in paragraaf 3.12. Meetkundig gezien is een stuk schroefdraad allerminst een

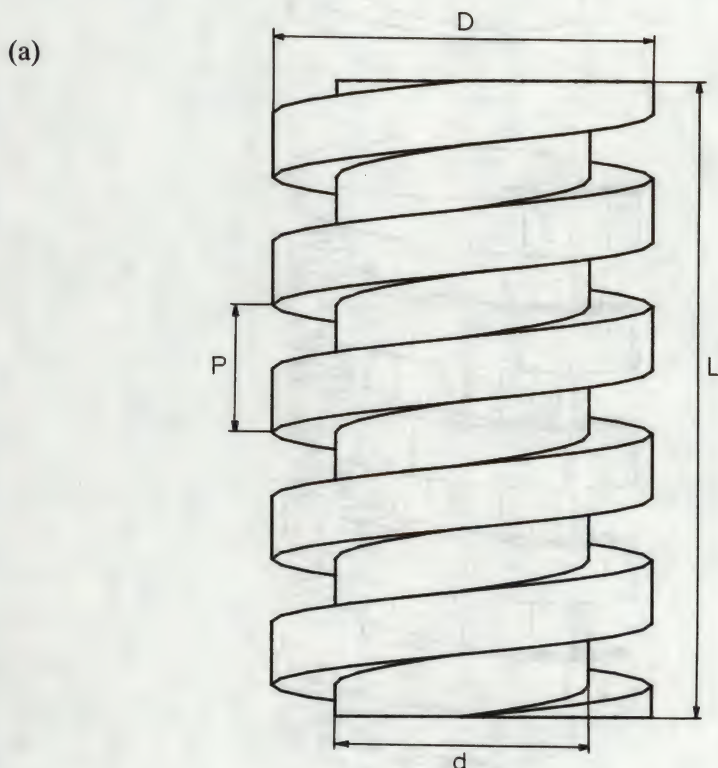
eenvoudig lichaam en programma SCRTHR zal dan ook niet zo gemakkelijk te begrijpen zijn. Maar gelukkig is het erg gemakkelijk in het gebruik. Het vraagt om drie belangrijke maten, namelijk een grote diameter D, een kleine diameter d, een steek P (van 'pitch') en een gewenste lengte L_0, aangegeven in figuur 2.21(a). Als de schroefdraad werkelijk vierkant is, dan heeft de zijde van zo'n denkbeeldig vierkant de lengte

$$0.5(D - d)$$

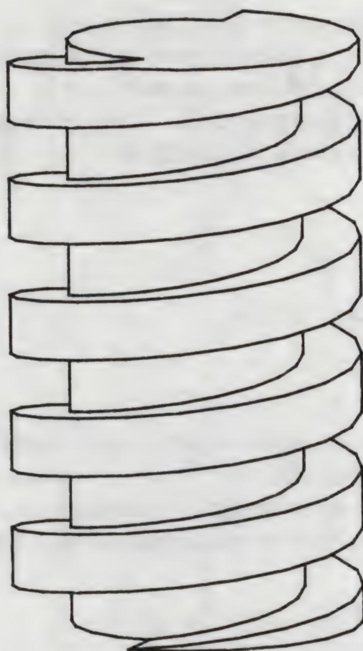
wat gelijk is aan de halve steek, zodat

$$P = D - d.$$

Als de schroefdraad dus echt vierkant is dan volgt de steek uit de grote en kleine diameter, zodat hij eigenlijk niet apart hoeft te worden opgegeven. Programma SCRTHR is wat algemener en vraagt om de steek, onafhankelijk van de twee diameters; op deze manier kunnen we ook schroefdraad verkrijgen dat niet vierkant maar rechthoekig is.



(b)



(c)



Figuur 2.21 Vierkant schroefdraad

Zoals gebruikelijk, worden ook hier gekromde oppervlakken benaderd door platte veelhoeken. We moeten daarom ook k opgeven, dat is het aantal stappen in een *halve* omwenteling. In paragraaf 2.2 hebben we het grondvlak van een cilinder benaderd door een regelmatige veelhoek met 30 zijden. Voor een soortgelijke benadering zouden we hier $k = 15$ moeten kiezen, want dan zouden we 30 stappen in een hele omwenteling hebben. Omdat schroefdraad veel complexer is dan een cilinder, kunnen we beter beginnen met een wat kleinere waarde, bijvoorbeeld $k = 8$, en die later verhogen, als dat gewenst blijkt te zijn.

Als de schroef een volledige omwenteling maakt, dan beweegt hij zich axiaal, dat wil zeggen in de richting van de as, over een afstand P, de steek. Omdat een volledige omwenteling uit $2k$ stappen bestaat, is de axiaal afgelegde afstand per stap gelijk aan

$$delta\ L = P / 2k.$$

De werkelijke lengte L wordt nu bepaald als dat gehele veelvoud van *delta L* dat het dichtst ligt bij de gewenste lengte L_0, zodat het verschil tussen de werkelijke en de gewenste lengte kleiner zal zijn dan de helft van *delta L*. Het is duidelijk dat deze twee lengten precies aan elkaar gelijk zijn als L_0 al een veelvoud van *delta L* is. Als we bijvoorbeeld hebben: $L_0 = 1.5$, $P = 0.25$ en $k = 6$, dan is

$$delta\ L = 0.25 / (2 \times 6) = 1/48$$

en

$$L_0 = 1.5 = 72 \times (1/48),$$

dus in dit geval is de gewenste lengte L_0 al een veelvoud van *delta L*, waardoor de werkelijke lengte L gelijk zal zijn aan L_0.

Afgezien van de maten, die de betekenis van D, d, P en L aangeven, zijn de figuren 2.21(a) en (b) vervaardigd door D3D, waarbij uitgegaan is van een objectfile gemaakt met programma SCRTHR. De volgende demonstratie van laatstgenoemd programma produceert die objectfile:

```
SCRTHR
Grote diameter D: 3
Kleine diameter d: 2
Bij echt vierkant draad hebben we:

pitch = D - d = 1.000
```

```
In een volledige omwenteling is 'pitch' de afstand
die de schroef voorwaarts beweegt, dus het 'vierkant'
heeft zijden met lengte

   h = pitch/2 = 0.500

Wilt u de steek (pitch) anders kiezen, zodat de draad
niet echt vierkant (maar rechthoekig) is? (J/N): N
Gewenste lengte van de schroefdraad
(niet kleiner dan 0.500): 5
Aantal stappen in een halve omwenteling: 10
De werkelijke lengte wordt: 5.000
Naam van uitvoerfile: FIG2.21
```

Als we de vraag over de steek met *J* hadden beantwoord, dan zou de volgende regel verschenen zijn:

```
Pitch:
```

We zouden dan een steek hebben kunnen opgeven die onafhankelijk is van de grote en kleine diameter.

Na op deze manier de file *FIG2.21* te hebben gegenereerd, kunnen we programma D3D starten door te typen

```
D3D FIG2.21
```

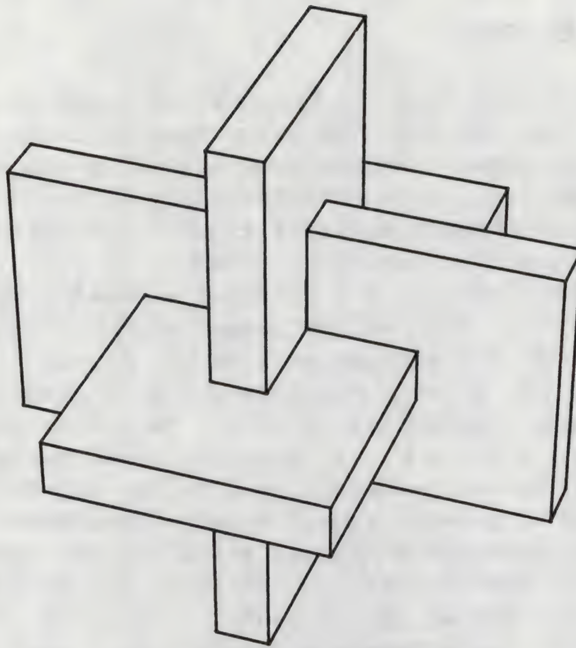
Bij een gecompliceerd voorwerp zoals dit stuk schroefdraad zijn we waarschijnlijk alleen geïnteresseerd in een perspectivisch beeld dat geen verborgen lijnen laat zien; in een geval als dit kunnen we het best op het werkscherm met alleen dunne lijnen werken. Ook zullen we waarschijnlijk willen dat de grensvlakken gekromde oppervlakken benaderen, waarbij de standaard drempelwaarde van 35° toegepast kan worden. De figuren 2.21(a) en (b) zijn op deze manier gemaakt, waarbij het oogpunt als volgt werd gekozen:

(a) $\rho = 1000$, $\theta = 0°$, $\varphi = 90°$
(b) $\rho = 1000$, $\theta = 20°$, $\varphi = 80°$.

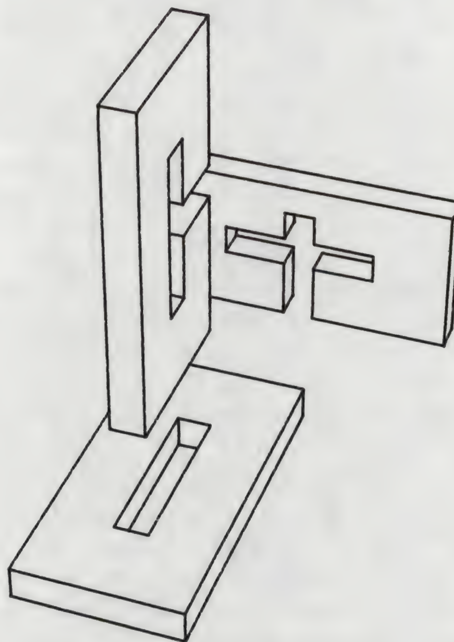
Figuur 2.21(c), verkregen met $\alpha = 0°$, toont hoe platte grensvlakken zijn gebruikt om gekromde oppervlakken te benaderen. We zullen dit in paragraaf 3.11 meer in detail bespreken.

2.7 'EXPLODED VIEWS'

Naast 'cutaway views', zie figuur 2.11, kunnen we voor ingewikkelde voorwerpen ook *exploded views* gebruiken. Deze kunnen nuttig zijn als het af te beelden voorwerp uit verschillende onderdelen bestaat en we willen laten zien hoe het in elkaar gezet moet worden. Als bijvoorbeeld gegeven is dat het voorwerp in figuur 2.22 uit drie onderdelen bestaat, dan zal het niet duidelijk zijn welke die onderdelen zijn, laat staan hoe we ze in elkaar kunnen zetten. Hiervoor gebruiken we liever de exploded view van figuur 2.23. Het laat de drie onderdelen duidelijk zien en suggereert dat we twee ervan aan een translatie moeten onderwerpen om het voorwerp van figuur 2.22 te verkrijgen, wat inderdaad het geval is.
Als u dat wilt kunt u deze twee perspectivische afbeeldingen met D3D vervaardigen, zoals ik dat zelf ook heb gedaan. Als buitenmaten van de drie onderdelen heb ik 40 x 20 x 4 (mm) gebruikt; de andere afmetingen volgen op eenvoudige wijze hieruit. Aanvankelijk waren de drie onderdelen in dezelfde positie, met hun centrum in de oorsprong van het coördinatenstelsel. Ik heb voor elk onderdeel een aparte objectfile gebruikt, maar ik kon de twee onderdelen die in figuur 2.23 op de voorgrond staan van de derde (die iets ingewikkelder is) afleiden door enkele hoekpunten weg te laten en de verzameling grensvlakken iets te wijzigen. Daarna was het niet moeilijk met behulp van enkele rotaties figuur 2.22 te verkrijgen. Tenslotte leverden enkele translaties figuur 2.23 op.



Figuur 2.22 Voorwerp bestaande uit drie onderdelen



Figuur 2.23 Exploded view

3 C-PROGRAMMA'S VOOR HET GENEREREN VAN D3D-INVOER

3.1 FUNCTIEPROTOTYPES IN TURBO C

Tot nu toe hebben we ons bij de bespreking van onze grafische programmatuur gesteld op het standpunt van de gebruiker. Van nu af aan zullen we ons voornamelijk bezighouden met programmeeraspecten, zodat de rest van het boek wellicht wat moeilijk leesbaar zal zijn als u niet bekend bent met programmeren. Maar zelfs in dat geval bent zult u het misschien kunnen waarderen dat de letterlijke tekst van alle besproken programma's in dit boek is opgenomen. U bent daardoor, als u de programma's gewijzigd zou willen hebben, niet afhankelijk van de auteur, maar u zou daarvoor bij elke ervaren C-programmeur kunnen aankloppen. Aan de andere kant, als u juist hoofdzakelijk in programmeeraspecten geïnteresseerd bent, dan moet toch ook aandacht besteden aan de hoofdstukken 1 en 2. Het heeft tenslotte geen zin te bespreken *hoe* een programma iets doet als u niet eens weet *wat* het doet.

Het zou mooi zijn als het programmeren net zo machine-onafhankelijk was als de wiskunde, want dan zouden al onze programma's altijd bruikbaar blijven. Het is verstandig machine- en compilerafhankelijke zaken te vermijden waar we dat maar kunnen, maar ongelukkigerwijze gaat dat soms niet. Dit laatste geldt in het bijzonder voor interactieve grafische software, die immers nogal nauw in verband staat met de gebruikte hardware. Niet alle onderwerpen in dit boek zijn systeemonafhankelijk. Met de nogal vage term 'systeem' duiden we hier de combinatie aan van de computer en de compiler die we gebruiken. We zullen C-programma's presenteren en bespreken die werken op de IBM PC en op de IBM PS/2. In de gebruikte machine, die natuurlijk ook een andere (of geen enkele) merknaam mag dragen mits hij met de genoemde machines compatibel is, moet een color graphics adapter (CGA), een enhanced graphics adapter (EGA/VGA) of een Hercules graphics adapter (HGA) aanwezig zijn. De hierbij gebruikte resoluties zijn respectievelijk 640 x 200, 640 x 350 en 720 x 348. Als u apparatuur gebruikt die wezenlijk hiervan verschilt dan zult u waarschijnlijk de programma's in dit boek niet in ongewijzigde vorm kunnen gebruiken. Toch zullen veel ideeën en basisprincipes dan ook op uw situatie van toepassing zijn, dus u kunt dan de hier besproken implementatie op de IBM PC beschouwen als niet meer dan een voorbeeld van hoe een en ander te realiseren is.

We zullen ons in dit boek houden aan Turbo C van Borland International. Als u een andere C-compiler voor de IBM PC wilt gebruiken dan zullen waarschijnlijk enkele 'laag-niveau' (dat wil zeggen 'machinegerichte') routines gewijzigd moeten worden, maar het is zeer onwaarschijnlijk dat zulke wijzigingen ernstige problemen zullen opleveren.

Dit boek gebruikt de programmeertaal C zonder die taal uit te leggen. Als u er meer van wilt weten dan raad ik u aan een van de boeken over C te raadplegen die in de literatuurlijst achterin dit boek worden genoemd.

De taal C is van oudsher berucht om zijn valkuilen. Dit komt voornamelijk doordat typecontrole en typeconversie achterwege blijft in sommige situaties waar we die controle en conversie wel verwachten. Als bijvoorbeeld de eerste regel van een functiedefinitie luidt

```
double f(x) double x;
```

dan is de aanroep *f*(1) incorrect omdat het integer argument niet geconverteerd wordt naar het dubbele-precisie floating-point type. De aanroep *f*(1.0) zou correct geweest zijn, aangezien de constante 1.0, op deze manier geschreven, het vereiste type heeft. Wat de zaak nog erger maakt, er wordt in dit geval geen typecontrole uitgevoerd, dus de compiler zal genoemde incorrecte aanroep accepteren, waardoor we pas tijdens de uitvoering van het programma (aan de onjuiste uitkomst) merken dat er iets mis is. Om dit soort narigheid, waarvan vooral beginners last hebben, het hoofd te bieden heeft het ANSI C Standards Committee een taalwijziging voorgesteld waarbij parametertypes op een zodanige manier worden gespecificeerd dat de compiler de gewenste controles en conversies kan uitvoeren. Omdat dit een afwijking is van de oorspronkelijke taal C, gedefinieerd door Kernighan & Ritchie (K & R) in *The C Programming Language*, lijkt het gevaar aanwezig dat er portabiliteitsproblemen ontstaan. Nu is het gelukkig zo dat veel compilers zowel de stijl van K & R als de voorgestelde ANSI-standaard accepteren. Een voorbeeld van zo'n compiler is Turbo C. In de User's Guide van deze compiler wordt veel aandacht besteed aan deze beide programmeerstijlen. In de terminologie van Borland zeggen we dat de oorspronkelijke K & R programma's geschreven zijn in de *klassieke* stijl, die we onderscheiden van de *moderne* stijl afkomstig van ANSI. Omdat ik een groot bewonderaar van K & R ben, doet deze terminologie mij prettig aan. De originele taal C verdient de positieve kwalificatie 'klassiek' en ik zou het betreurd hebben als in plaats hiervan het woord 'ouderwets' gebruikt was. Aan de andere kant moeten we onze ogen niet sluiten voor het feit dat de taal C erg populair aan het worden is, wat betekent dat een groot aantal beginners ermee bezig zijn, voor wie een beetje hulp van de compiler bij de controle op correctheid uiterst nuttig is. Laten we een voorbeeld gebruiken om te zien waarover we het precies hebben. In de klassieke stijl kunnen we schrijven:

```
/* KLASSIEK */
double f();               /* Declaratie van f */

main()
{ printf("%3.1f", f(2.0));
}

double f(x) double x;  /* Definitie van f  */
{ return 3.0 * x;
}
```

We mogen in dit programma 2.0 niet vervangen door 2, want dat zou niet de gewenste uitvoer 6.0, maar een of andere incorrecte waarde opleveren. In de moderne stijl kunnen we schrijven:

```
/* MODERN */
double f(double x);     /* Functieprototype */

main()
{ printf("%3.1f", f(2.0));
}

double f(double x)      /* Functiedefinitie */
{ return 3.0 * x;
```

De manier waarop functie *f* aan het begin van dit programma is gedeclareerd is een ANSI-taaluitbreiding, *functieprototype* geheten. Deze vertelt de compiler niet alleen (zoals een functiedeclaratie in de klassieke stijl dat doet) dat functie *f* een waarde van het type *double* teruggeeft, maar ook dat een argument van dat type wordt verwacht. Een compiler (die de moderne stijl accepteert) zal dit controleren en, als het argument een ander type heeft, ofwel een foutmelding geven ofwel het gegeven type converteren naar het type dat vereist is.

Turbo C accepteert de beide bovenstaande programma's *KLASSIEK* en *MODERN*. Als we in het laatstgenoemde *f*(2) schrijven dan zal *int* waarde 2 automatisch worden geconverteerd naar het type *double*, alsof we 2.0 hadden geschreven. Misschien is door dit uiterst eenvoudige voorbeeld het belang van functieprototypes nog niet helemaal duidelijk. We moeten ons evenwel realiseren dat in grote en complexe programma's, zoals we die in dit boek zullen bespreken, het vinden van programmeerfouten op het gebied van typeconversie ons uren kan kosten als de compiler ons hierbij niet behulpzaam is.

Grote programma's worden gewoonlijk verdeeld in verschillende modulen. Als we zo'n modulaire aanpak toepassen op ons eenvoudige voorbeeld, dan krijgen we de volgende situatie:

Module 1:

```
/* MODERN */
double f(double x);  /* Functieprototype */

main()
{ printf("%3.1f", f(2.0));
}
```

Module 2:

```
double f(double x)
{ return 3.0 * x;
}
```

Wanneer de compiler met module 1 bezig is, zou hij zonder het functieprototype niet over informatie beschikken die vermeld is in de definitie van functie *f* (dat wil zeggen in de functie zelf), dus dit is een duidelijker geval waarin een functieprototype nuttig is. Let er ook op dat het de moeite waard is, niet alleen de *types* van de functie-argumenten maar ook hun *aantal* te controleren. Als we in module 1 *f*(2.0) vervangen door *f*(), of door *f*(2.0, 3.0, 4.0), dan zal de compiler een foutmelding geven, omdat het functieprototype zegt dat er precies één argument moet zijn. Dit zou niet mogelijk zijn als we het functieprototype hadden vervangen door de regel

```
double f();
```

wat we volgens de klassieke stijl zouden schrijven. In dat geval zou de compiler niet klagen, maar zouden er ernstige problemen ontstaan bij de uitvoering van het gecompileerde (en 'gelinkte') programma.

Gebruikers van Turbo C Versie 1.0 die programmeren in de klassieke stijl worden hiervoor gestraft als zij niet erg zorgvuldig de types *float* en *double* van elkaar onderscheiden. Traditioneel worden argumenten van het type *float* geconverteerd ('widened') naar het type *double*. Op deze manier wordt een eventuele incompatibiliteit tussen *float* en *double* opgelost, want, zelfs als parameters als *float* worden gespecificeerd zal de functie in kwestie argumenten van het type *double* verwachten. De normale conventie is dus dat intern zowel de (actuele) argumenten als de (formele) parameters altijd het type *double* hebben wanneer hun type

floating-point is. Ongelukkigerwijs wijkt Turbo C Versie 1.0 van dit principe af en neemt de types van floating-point parameters en argumenten heel letterlijk. Dit betekent dat we in de klassieke stijl in moeilijkheden komen als we een parameter van het type *float* combineren met een floating-point *constante*, want zulke constanten (zoals bijvoorbeeld 2.0) zijn altijd van het type *double*. Met de moderne stijl hebben we deze problemen niet, want hier schrijven de functieprototypes voor hoe argumenten moeten worden geconverteerd naar het type van de overeenkomstige parameters. Dit leek mij een van de belangrijkste argumenten ten gunste van de moderne stijl, vooral omdat ik toen nog niet wist dat dit alleen maar een tijdelijk probleem was. Toen ik Turbo C Versie 1.5 ontving (zie ook paragraaf 4.3) bleek dat de compiler op dit punt gecorrigeerd was, zodat aanhangers van de klassieke stijl nu gerust *double* argumenten kunnen combineren met *float* parameters en vice versa. Voor (de vermoedelijk vele) gebruikers die Versie 1.0 nog niet door een nieuwere hebben vervangen, is de moderne stijl om bovengenoemde redenen nog steeds dringend aan te bevelen. Verder blijkt dat de moderne stijl (bij gebruik van Versie 1.5) efficiëntere objectcode aflevert dan de klassieke stijl, omdat bij gebruik van de moderne stijl de argumenten niet eerst van *float* naar *double* en daarna weer naar *float* geconverteerd hoeven te worden. U kunt het verschil zelf constateren als u de volgende twee functies afzonderlijk compileert:

```
void f(float x) { }     /* Modern   */

void f(x) float x; { }  /* Klassiek */
```

Als we met Turbo C Versie 1.5 deze twee modules afzonderlijk compileren, waarbij we 8088-code genereren voor het 'huge' geheugenmodel, dan zijn de lengten voor deze modules respectievelijk 143 en 188 bytes, dus de moderne stijl leidt tot compactere code.

Onze bespreking van functieprototypes is nog niet compleet. Ten eerste kunnen de namen van parameters worden weggelaten, dus in plaats van

```
double f(float x);
```

mogen we ook schrijven:

```
double f(float);
```

Eerstgenoemde vorm zal toch vaak de voorkeur verdienen, omdat de naam van de parameters ons vaak een idee geeft wat de functie doet. Ten tweede kan het keyword *void* nuttig zijn. Als functie *f* in het geheel geen parameters heeft dan kunnen we schrijven

```
double f(void);
```

Deze regel zal ervoor zorgen dat de compiler een foutboodschap geeft als we *f* aanroepen met één of meer argumenten. Als we, in plaats van het bovenstaande, de klassieke declaratie

```
double f();
```

schrijven, dan zal de compiler elk aantal argumenten toestaan en ten aanzien van dit aantal geen fouten rapporteren, dus het keyword *void* is hier echt nuttig.

We hebben het keyword *void* ook nodig om aan te geven dat een functie geen waarde aflevert. We schrijven dan *void* (in plaats van bijvoorbeeld *double*) voorafgaande aan de functienaam, zoals in:

```
void p(int i);

main()
{ p(3);
}

void p(int i)
{ printf("%d", i);
}
```

Laten we *void* in de eerste regel (het functieprototype) weg, dan zal dat leiden tot een foutmelding. Als we het woord *void* in deze regel wel opnemen dan moeten we het ook in de functiedefinitie zelf gebruiken, want anders zou de compiler aannemen dat de functie een *int* waarde teruggeeft (zoals dat vereist wordt in de originele taal, zie K & R), wat in strijd zou zijn met het functieprototype. Dit betekent dat we in dit programma het tweede gebruik van *void* achterwege zouden kunnen laten, mits we tegelijkertijd in het functieprototype het woord *void* vervangen door het woord *int*. Omdat functie *p* geen return-statement bevat is bovenstaande oplossing veel mooier.

Turbo C staat ons toe een functiedefinitie in klassieke stijl ook te gebruiken in combinatie met een functieprototype, zodat we bijvoorbeeld kunnen schrijven:

```
int f(float x);   /* Modern functieprototype    */
...
int f(x) float x; /* Klassieke functiedefinitie */
{ ...
}
```

In dit geval wordt controle en conversie van de argumenten op dezelfde wijze uitgevoerd als in:

```
int f(float x);    /* Modern functieprototype   */
...
int f(float x)     /* Moderne functiedefinitie  */
{ ...
}
```

Het vermengen van de klassieke en de moderne stijl is niet bijzonder elegant. Als we de moderne stijl consequent toepassen, dan kunnen we eenvoudig ieder functieprototype schrijven als een exacte kopie van de eerste regel van de corresponderende functiedefinitie, met uitzondering van de puntkomma, die, aan het eind van de regel, wel voorkomt in het functieprototype maar niet in de functiedefinitie. We geven daarom de voorkeur aan de tweede van bovenstaande twee vormen, en zullen van nu af aan consequent de moderne stijl gebruiken.

We dienen ons ervan bewust te zijn dat we in Turbo C functieprototypes gebruiken zodra we gebruik maken van standaard header-files, zoals bijvoorbeeld de bekende file *math.h*, die in ons programma wordt opgenomen door de regel

```
#include <math.h>
```

Bij compilers die alleen de klassieke stijl kennen, wordt bijvoorbeeld de functie *atan* in deze header-file gedeclareerd als

```
double atan();
```

maar in Turbo C vinden we daarvoor in de plaats:

```
double atan(double x);
```

Dit betekent dat door bovengenoemde *#include* regel dit functieprototype, evenals vele andere, in ons programma wordt opgenomen, zonder dat we daar misschien erg in hebben. Wat de functie *atan* betreft betekent dit dat we kunnen schrijven

```
pi = 4.0 * atan(1);
```

in plaats van

```
pi = 4.0 * atan(1.0);
```

om de waarde π te benaderen. Toch gebruiken we om redenen van portabiliteit liever de tweede vorm dan de eerste.

We kunnen het als een nadeel beschouwen dat onze programma's door het gebruik van de moderne stijl niet geaccepteerd worden door compilers die alleen de

klassieke stijl kennen. Zelf ben pas ik op de moderne stijl overgestapt na de zaak van alle kanten bekeken te hebben. Tenslotte heb ik in vijf boeken (één in het Nederlands en vier in het Engels) overal de klassieke stijl gebruikt en het is jammer deze uniformiteit van stijl te moeten prijsgeven. Ik heb daarom eerst geprobeerd bij de klassieke stijl te blijven, maar lettend op alles wat we in deze paragraaf hebben besproken leek het me toch beter op de moderne stijl over te stappen. Als u hetzelfde wilt doen dan zou ik u willen adviseren consequent te zijn. Ook hierbij kan Turbo C ons helpen, namelijk door foutboodschappen te geven als we een functieprototype vergeten. In de 'geïntegreerde omgeving' (waarbij we voor de compilatie en de uitvoering van het programma de editor niet verlaten) kunnen we dit doen door eerst *Options*, daarna *Compilers*, dan *Errors* en tenslotte *Less common errors* te selecteren; hierna zetten we de opties *Call to function with no prototypes* en *No declaration for function* aan. Tussen twee haakjes, het is ook verstandig de meeste andere opties van de menu's *Less common errors* en *Common errors* aan te zetten, want daar zitten verscheidene nuttige mogelijkheden bij zoals bijvoorbeeld het signaleren dat een variabele wel gedefinieerd maar nooit gebruikt wordt.

Een functieprototype is in feite een functiedeclaratie die vollediger is dan zo'n declaratie in de klassieke stijl. Dit betekent dat we, net als klassieke functiedeclaraties, een aantal ervan kunnen samenvoegen en ze zelfs kunnen combineren met programmavariabelen zoals *x* in:

```
float x, f(int i, char ch), g(void);
```

Deze regel is equivalent met de volgende drie regels:

```
float x;
float f(int i, char ch);
float g(void);
```

Het is ook nuttig te weten dat een apart functieprototype overbodig is als, binnen één programmamodule, alle aanroepen naar een functie pas voorkomen *na* de definitie van die functie, aannemende dat deze definitie volgens de moderne stijl is geschreven. Het volgende voorbeeld laat dit zien:

```
float f(float x) /* Functiedefinitie in moderne stijl */
{ return x * x + x + 1;
}

main()
{ printf("%f", f(1))
}
```

Dank zij de moderne stijl wordt het *int* argument 1 geconverteerd naar het type *float*, hoewel er niet een apart functieprototype voor *f* is. De eerste regel van functie *f* fungeert hier in feite als functieprototype en we zullen hem daarom als zodanig beschouwen.

Zojuist is geadviseerd de compiler te instrueren boodschappen te geven als functieprototypes ontbreken. Als u dit advies opvolgt dan moet u uiteraard die prototypes in uw programma opnemen. Dit is ook van toepassing op 'standaardfuncties', zoals *printf* en *exit.* De prototypes voor zulke functies staan in header-files. Daarom zullen we bijvoorbeeld in elk programma dat gebruik maakt van *printf* (en er zijn weinig programma's die dat niet doen) de regel

```
#include <stdio.h>
```

bovenaan opnemen. Analoog hieraan is de functie *exit* gedeclareerd in de header file *process.h.* Het zal nu duidelijk zijn waarom er in de programma's in dit boek zo veel include-regels voorkomen. Zoals u waarschijnlijk weet, hebben we in de klassieke stijl zo'n include-regel niet nodig voor standaardfuncties die een waarde van het type *int* (of helemaal geen waarde) afleveren. Zulke include-regels zijn nu wel nodig (of althans gewenst) maar dit extra werk is welbesteed, in het bijzonder als we soms lijden aan het menselijke euvel van slordigheid. Zo zal bijvoorbeeld de incorrecte statement

```
printf('A');
```

bij Turbo C een duidelijke foutmelding geven als we bovenaan in ons programma de regel

```
#include <stdio.h>
```

opnemen. Zonder deze regel (en werkend in de klassieke stijl) zou deze ernstige fout ('A' in plaats van "A") tijdens programma-uitvoering een bijzonder vervelend gevolg hebben, omdat dan de ASCII-waarde van 'A' (dat wil zeggen 65) gebruikt zou worden als het beginadres van een 'string' die wordt afgedrukt.

Dit is het einde van een beschouwing over Turbo C die u misschien erg langdradig zult vinden. Omdat er zoveel programmatekst in dit boek voorkomt leken mij de hier genoemde taalaspecten toch belangrijk genoeg om er vrij diep op in te gaan voordat we ons gaan verdiepen in de programma's zelf, waarvoor u misschien meer belangstelling hebt.

3.2 CILINDERS EN PRISMA'S

We zullen beginnen met enkele vrij eenvoudige programma's; deze produceren files die door D3D worden gelezen. Het eerste programma is CYLINDER. We hebben in paragraaf 2.2 de gebruikersaspecten van dit programma al behandeld en we zullen nu de programmatekst eens bekijken, zodat u daarna ook zelf zulk soort programma's kunt maken.

```
/* CYLINDER: Het genereren van een cilinder
*/
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <process.h>
main()
{ FILE *fp;
  char ch, str[50];
  float diam, r, h, delta, alpha, pi, rcos, rsin, half;
  int i, i1, n;
  pi = 4*atan(1.0);
  do
  { printf("Richting van de cilinderas? (X/Y/Z) ");
    ch = getchar(); ch = toupper(ch);
  } while (ch != 'X' && ch != 'Y' && ch != 'Z');
  printf("Diameter? "); scanf("%f", &diam); r = diam/2;
  printf("Hoogte? "); scanf("%f", &h); half = h/2;
  printf("Aantal hoekpunten in regelmatige veelhoek? ");
  scanf("%d", &n);
  printf("Naam van de uitvoerfile? "); scanf("%s", str);
  fp = fopen(str, "w");
  if (fp == NULL) {printf("File-probleem"); exit(1);}
  delta = 2 * pi / n;
  for (i=1; i<=n; i++)
  { alpha = i * delta;
    rcos = r * cos(alpha); rsin = r * sin(alpha);
    if (ch == 'Z')
    { fprintf(fp, "%d %f %f %f\n", i, rcos, rsin, half);
      fprintf(fp, "%d %f %f %f\n", i+n, rcos, rsin, -half);
    } else
    if (ch == 'X')
    { fprintf(fp, "%d %f %f %f\n", i, half, rcos, rsin);
      fprintf(fp, "%d %f %f %f\n", i+n, -half, rcos, rsin);
    } else  /* ch == 'Y' */
    { fprintf(fp, "%d %f %f %f\n", i, rsin, half, rcos);
      fprintf(fp, "%d %f %f %f\n", i+n, rsin, -half, rcos);
    }
```

```
  }
  fprintf(fp, "Faces:\n");
  for (i=1; i<=n; i++)
    fprintf(fp, "%d%s", i,
    i == n ? ".\n" : i%10 ? " " : "\n");
  for (i=1; i<=n; i++)
    fprintf(fp, "%d%s", 2*n+1-i,
    i == n ? ".\n" : i%10 ? " " : "\n");
  for (i=1; i<=n; i++)
  { i1 = i==n ? 1 : i+1;
    fprintf(fp, "%d %d %d %d.\n", i, i+n, i1+n, i1);
  }
  fclose(fp);
}
```

Zoals gewoonlijk gebruiken we als grondvlak een regelmatige veelhoek in plaats van een cirkel. Dit betekent dat we dit programma niet alleen voor een cilinder maar ook voor een prisma kunnen gebruiken, althans als het grondvlak ervan een regelmatige veelhoek is. Dit wordt gedemonstreerd in figuur 3.1, waarin ook te zien is hoe de hoekpunten zijn genummerd.

Een programmastatement zoals

```
fprintf(fp, "%d%s", i,
i == n ? ".\n" : i%10 ? " " : "\n");
```

zal misschien niet onmiddellijk duidelijk zijn. Hij bevat een geneste conditionele expressie van de vorm

ppp ? *aaa* : *qqq* ? *bbb* : *ccc*

waarin de operatoren van rechts naar links associëren, wat betekent dat we dit moeten opvatten als:

ppp ? *aaa* : (*qqq* ? *bbb* : *ccc*)

Met wat meer programmatekst kunnen we ook gebruik maken van een conditionele statement en schrijven:

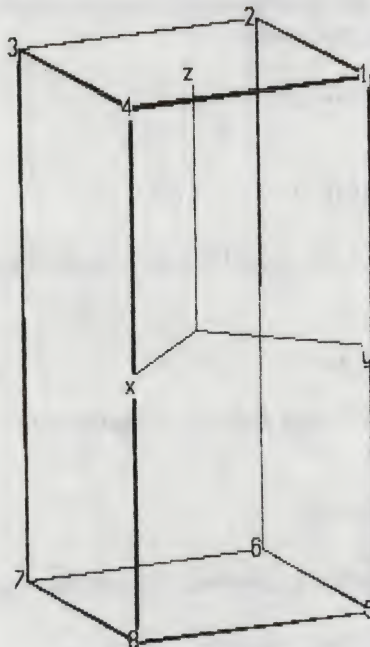
```
if (i == n) fprintf(fp, "%d.\n", i); else
if (i % 10 != 0) fprintf(fp, "%d ", i); else
                 fprintf(fp, "%d\n", i);
```

We geven dus nadat *i* is afgedrukt een punt en een newline-karakter als *i* gelijk is aan *n*; als dat niet zo is geven we een spatie, behalve als *i* een veelvoud van 10 is, in welk geval we een newline-karakter (in plaats van de spatie) geven. Zowel in de oorspronkelijke als in deze uitgebreidere vorm zullen ten hoogste tien hoekpuntnummers op een regel verschijnen.

De rest van de programmatekst zal vermoedelijk duidelijk zijn als u figuur 3.1 erbij betrekt. We zullen in dit boek een groot aantal van dit type programma's onder de loep nemen en mogelijk zult u ze ook zelf gaan schrijven. Voor al die programma's geldt dat allereerst de puntnummering volkomen duidelijk zijn.

Het prisma van figuur 3.1 kan als volgt worden verkregen:

```
cylinder
Richting van de cilinderas? (X/Y/Z) z
Diameter? 10
Hoogte? 15
Aantal hoekpunten in regelmatige veelhoek? 4
Naam van de uitvoerfile? fig3.1
```



Figuur 3.1 Prisma als uitvoer van programma CYLINDER

We typen dan

```
d3d fig3.1
```

en gebruiken commando *M* om op te geven dat we de hoekpuntnummers en de assen in de figuur willen zien en dat we het hele scherm willen gebruiken; daarna geven we commando *P* om het resultaat op de manier van figuur 3.1 te laten afdrukken. Let erop dat de zijden van het vierkante onder- en bovenvlak koorden zijn van de cirkel die we zouden benaderen als we een groot aantal hoekpunten hadden genomen. Bij deze vierkanten hebben de diagonalen (en niet de zijden) de lengte 10 van de hierboven genoemde 'diameter'.

Omdat de file FIG3.1 heel klein is kunnen we net zo goed de inhoud ervan hier letterlijk opnemen:

```
1 -0.000000 5.000000 7.500000
5 -0.000000 5.000000 -7.500000
2 -5.000000 -0.000000 7.500000
6 -5.000000 -0.000000 -7.500000
3 0.000000 -5.000000 7.500000
7 0.000000 -5.000000 -7.500000
4 5.000000 0.000001 7.500000
8 5.000000 0.000001 -7.500000
Faces:
1 2 3 4.
8 7 6 5.
1 5 6 2.
2 6 7 3.
3 7 8 4.
4 8 5 1.
```

3.3 KEGELS EN PIRAMIDEN

Programma CONE produceert een kegel die benaderd wordt door een piramide, wat analoog is aan onze benadering van een cilinder door een prisma in de vorige paragraaf:

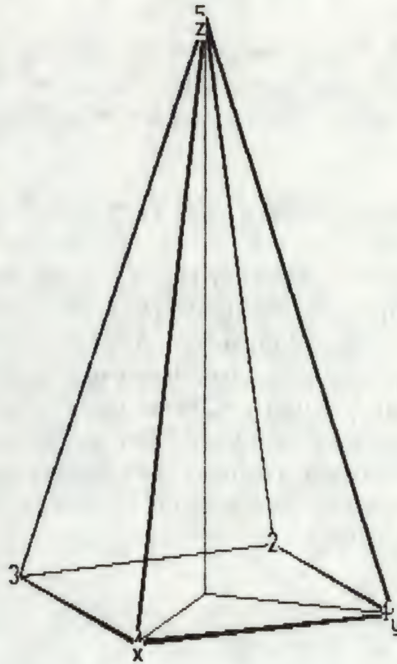
```
/* CONE: Het genereren van een kegel
*/
#include <math.h>
```

```
#include <stdio.h>
#include <process.h>
main()
{ FILE *fp;
  int n, i;
  float diam, r, h, delta, alpha, x, y, pi;
  char str[50];
  pi = 4.0 * atan(1.0);
  printf("Diameter van grondvlak? "); scanf("%f", &diam);
  r = diam/2;
  printf("Hoogte? "); scanf("%f", &h);
  printf("Aantal hoekpunten van veelhoek? ");
  scanf("%d", &n);
  printf("Naam van de uitvoerfile? "); scanf("%s", str);
  delta = 2*pi/n;
  fp = fopen(str, "w");
  if (fp == NULL) {printf("File-probleem"); exit(1);}
  for (i=1; i <= n; i++)
  { alpha = i * delta;
    x = r * cos(alpha); y = r * sin(alpha);
    fprintf(fp, "%2d %f %f %f\n", i, x, y, 0.0);
  }
  fprintf(fp, "%2d %f %f %f\n", n+1, 0.0, 0.0, h);
  fprintf(fp, "Faces:\n");
  for (i=n; i>=1; i--) fprintf(fp, "%3d%s", i,
     (i==1 ? ".\n" : (i%10 == 1 ? "\n" : "")));
  for (i=1; i<=n; i++)
  fprintf(fp, "%2d %2d %2d.\n",
     i, (i == n ? 1 : i+1), n+1);
  fclose(fp);
}
```

De volgende demonstratie laat zien hoe we met dit programma een file kunnen verkrijgen die dan door programma D3D kan worden gelezen, waarbij figuur 3.2 ontstaat:

```
cone
Diameter van grondvlak? 10
Hoogte? 15
Aantal hoekpunten van veelhoek? 4
Naam van de uitvoerfile? fig3.2
```

Niet alle puntnummers zijn in figuur 3.2 duidelijk leesbaar. Dit komt doordat de letters *x*, *y* en *z* aan het eind van de drie positieve assen later dan die puntnummers



Figuur 3.2 Piramide als uitvoer van programma CONE

worden afgedrukt: op het beeldscherm overschrijven de x, y en z gedeeltelijk de nummers 4, 1 en 5. We laten niet vaak puntnummers afdrukken en als we dat wel doen dan kunnen we door middel van commando *M* ervoor zorgen dat de assen worden weggelaten; in dat geval blijven de letters x, y en z ook achterwege, zodat de puntnummers er niet door verminkt worden.

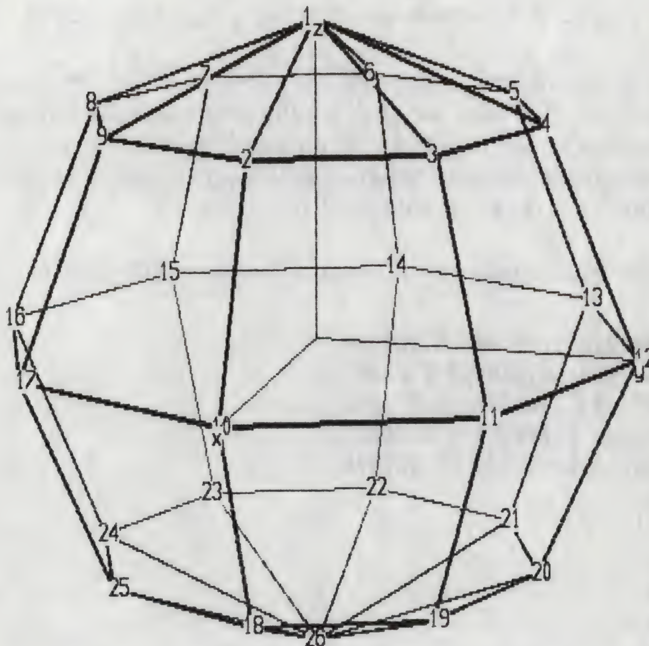
Terwille van de volledigheid volgt hier ook de inhoud van de file FIG3.2:

```
1 -0.000000 5.000000 0.000000
2 -5.000000 -0.000000 0.000000
3 0.000000 -5.000000 0.000000
4 5.000000 0.000001 0.000000
5 0.000000 0.000000 15.000000
Faces:
4 3 2 1.
1 2 5.
2 3 5.
3 4 5.
4 1 5.
```

Natuurlijk kunnen we deze eenvoudige file ook wel zonder programma CONE verkrijgen. Het nut van dit programma komt daarom beter tot uiting bij een meer realistische benadering van een kegel, zoals getoond in paragraaf 2.2.

3.4 TRADITIONELE BENADERING VAN EEN BOL

In paragraaf 2.2 hebben we op twee manieren een bol benaderd, namelijk met programma SPHERE en met de file SPH80.DAT. Met het programma SPHERE, dat het onderwerp van deze paragraaf is, maken we gebruik van m horizontale schijven tussen de twee polen en van n meridianen (dat zijn grote cirkels die door de polen gaan). In plaats van $m = 10$ en $n = 20$ te gebruiken, wat we deden in paragraaf 2.2, kunnen we beter een voorbeeld kiezen met veel kleinere waarden voor deze variabelen, zodat de puntnummers duidelijk in de figuur zichtbaar gemaakt kunnen worden. Met $m = 4$ en $n = 8$ geeft programma SPHERE, gevolgd door D3D, het resultaat dat te zien is in figuur 3.3.



Figuur 3.3 Bolbenadering met $m = 4$ en $n = 8$

De methode die we gebruiken berust op de bekende relatie tussen de rechthoekige coördinaten x, y en z enerzijds en de bolcoördinaten R, θ en φ anderzijds, voor alle punten op de bol met straal R en middelpunt O:

$$x = R \sin \varphi \cos \theta$$
$$y = R \sin \varphi \sin \theta$$
$$z = R \cos \varphi$$

Breedtecirkels (die in een horizontaal vlak liggen) worden verkregen door φ constant en θ variabel te nemen; op analoge wijze verkrijgen we met constante θ en variabele φ de meridianen.

In programma SPHERE gebruiken we een integer variabele i, die loopt van 0 tot m, en nummeren we de punten van boven naar beneden als volgt:

$i = 0$ (noordpool): 1
$i = 1$ (eerste breedtecirkel): 2, 3, ..., $n+1$
$i = 2$ (tweede breedtecirkel): $n+2$, $n+3$, ..., $2n+1$
$i = 3$ (derde breedtecirkel): $2n+2$, $2n+3$, ..., $3n+1$
. . .
$i = m$-1 (laatste breedtecirkel):
$(m\text{-}2)n+2$, $(m\text{-}2)n+3$, ..., $(m\text{-}1)n+1$
$i = m$ (zuidpool): $(m\text{-}1)n+2$

Als benaderende platte grensvlakken gebruiken we driehoeken bij de polen en vierhoeken overal elders. (Meer in het bijzonder zijn deze vierhoeken *trapezia*: zij hebben twee evenwijdige zijden.) Met al deze informatie (inclusief figuur 3.3) moet programma SPHERE duidelijk zijn:

```
/* SPHERE: Bolbenadering met m schijven en n
              punten op elke cirkel.
   */
   #include <stdio.h>
   #include <math.h>
   #include <process.h>
   main()
   { FILE *fp;
     int i, j, m, n, nr, next, southpole;
     double r, theta, phi, pi, delphi, deltheta,
       rsinphi, rcosphi, x, y, z;
     char str[30];
     pi = 4 * atan(1.0);
     printf("Geef m, het aantal horizontale schijven: ");
     scanf("%d", &m);
```

```
  printf(
"\nEr zullen n punten op elke horizontale cirkel liggen.\n");
  printf(
"Een waarde n = 2m wordt aanbevolen.");
  printf("\nGeef n: "); scanf("%d", &n);
  delphi = pi/m; deltheta = 2*pi/n;
  printf("Straal van de bol: "); scanf("%lf", &r);
  printf("Naam van de uitvoerfile: "); scanf("%s", str);
  fp = fopen(str, "w");
  if (fp == NULL) {printf("File-probleem"); exit(1);}
  /* Puntnummering:
     i=0:   1
     i=1:   2, 3, ..., n+1
     i=2:   n+2, n+3, ..., 2n+1
          ...
     i=m-1: (m-2)n+2, (m-2)n+3, ..., (m-1)n+1
     i=m:   (m-1)n+2
  */
  fprintf(fp, "%d %f %f %f\n", 1, 0.0, 0.0, r);  /* i = 0 */
  for (i=1; i<m; i++)
  { phi = i * delphi;
    rcosphi = r * cos(phi); rsinphi = r * sin(phi);
    for (j=0; j<n; j++)
    { nr = (i-1) * n + j + 2;
      theta = j * deltheta;
      x = rsinphi * cos(theta);
      y = rsinphi * sin(theta);
      z = rcosphi;
      fprintf(fp, "%d %f %f %f\n", nr, x, y, z);
    }
  }
  fprintf(fp, "%d %f %f %f\n", (m-1)*n+2, 0.0, 0.0, -r);
                                                  /* i = m */
  fprintf(fp, "Faces:\n");
  for (j=2; j<=n+1; j++)
    fprintf(fp, "%d %d %d.\n", 1, j, (j<n+1 ? j+1 : 2));
  for (i=1; i<m-1; i++)
  { nr = (i-1) * n;
    for (j=2; j<=n+1; j++)
    { next = (j < n+1 ? j+1 : 2);
      fprintf(fp, "%d %d %d %d.\n",
                  nr+j, nr+n+j, nr+n+next, nr+next);
    }
  }
```

```
    southpole = (m-1)*n+2; nr = (m-2)*n;
    for (j=2; j<=n+1; j++)
      fprintf(fp, "%d %d %d.\n",
                 nr+j, southpole, (j<n+1 ? nr+j+1 : nr+2));
    fclose(fp);
}
```

3.5 REGELMATIGE VEELVLAKKEN

Een veelvlak is een lichaam waarvan de grensvlakken veelhoeken zijn. Als alle zijden van een veelhoek even lang en alle hoeken even groot zijn wordt de veelhoek *regelmatig* genoemd. Evenzo zeggen we dat een *veelvlak regelmatig* is als al zijn grensvlakken identieke regelmatige veelhoeken zijn. Er zijn slechts vijf regelmatige veelvlakken, ook wel *Platonische lichamen* genoemd; zij zijn afgebeeld in figuur 3.4. Hun naam (Nederlands en internationaal) en hun voornaamste eigenschappen zijn hieronder genoemd:

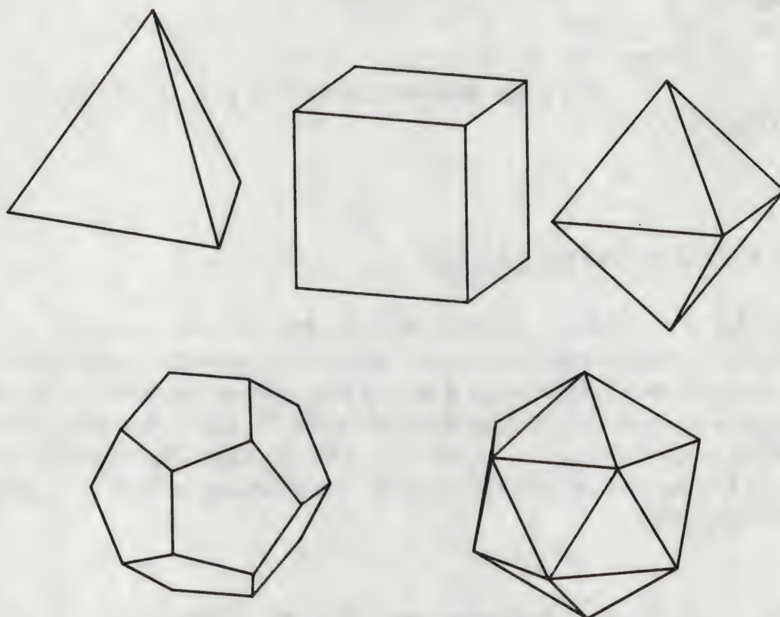
Veelvlak	Int. naam	Grensvlakken	Ribben	Hoekpunten
Vierhoek	Tetraëder	4	6	4
Zesvlak	Hexaëder	6	12	8
Achtvlak	Octaëder	8	12	6
Twaalfvlak	Dodecaëder	12	30	20
Twintigvlak	Icosaëder	20	30	12

Terloops dient te worden vermeld dat de hier genoemde getallen voldoen aan de stelling van Euler, die zegt dat voor elk veelvlak (al of niet regelmatig) de volgende gelijkheid geldt:

Grensvlakken + Hoekpunten = Ribben + 2

(Kortheidshalve zullen we het woord *regelmatig* vaak weglaten: met bijvoorbeeld een 'twintigvlak' zullen we een regelmatig twintigvlak bedoelen.)

Figuur 3.4 is gemaakt met behulp van programma D3D. We zien er de vijf regelmatige veelvlakken in dezelfde volgorde als in bovenstaande tabel. Voor het twaalfvlak en het twintigvlak heb ik afzonderlijke programma's gebruikt om de objectfiles te genereren; deze programma's zijn opgenomen in de paragrafen 3.5.4 en 3.5.5. Laten we nu de vijf regelmatige veelvlakken één voor één gaan bekijken.



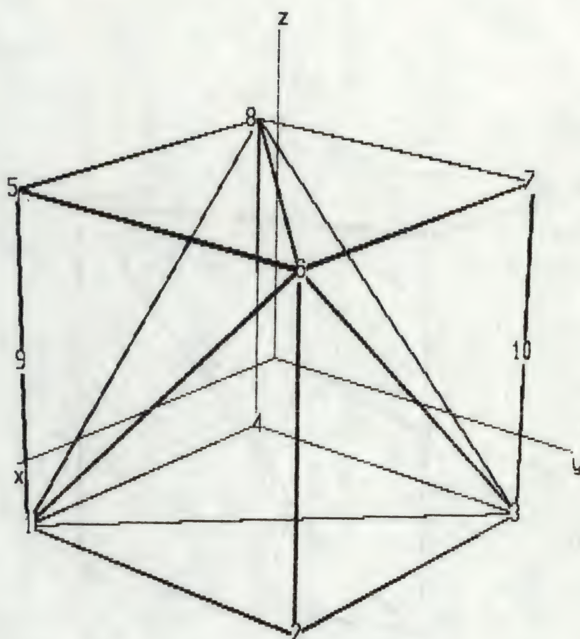
Figuur 3.4 De vijf regelmatige veelvlakken:
Viervlak, zesvlak, achtvlak;
twaalfvlak, twintigvlak

3.5.1 Viervlak

Hoewel een viervlak, met vier gelijkzijdige driehoeken als grensvlakken, erg eenvoudig lijkt, is het tekenen van een perspectivisch beeld ervan toch niet triviaal. De gemakkelijkste manier om dit lichaam te construeren is het af te leiden van een kubus, zoals figuur 3.5 laat zien. Met D3D gebruiken we commando *F*, gevolgd door de volgende vier regels om de begrenzende driehoeken te definiëren:

```
1 3 6.
3 8 6.
1 6 8.
1 8 3.
```

Doordat het viervlak geen horizontaal grondvlak heeft staat het niet in de gewenste stand. Gelukkig kunnen we (met commando *T*) het viervlak om een willekeurige as roteren om hierin verbetering te brengen. We zouden de ribbe 1-3 als zo'n as van rotatie kunnen gebruiken en het viervlak naar ons toe draaien zodat hoekpunt 6 omlaag gaat totdat het in het xy-vlak ligt; driehoek 1 3 6 zou dan horizontaal geworden zijn. In werkelijkheid heb ik gebruik gemaakt van de as door de punten 9 en 10, die ook in figuur 3.5 zijn aangegeven. Deze lijn gaat door de oorsprong O van



Figuur 3.5 Viervlak in een kubus

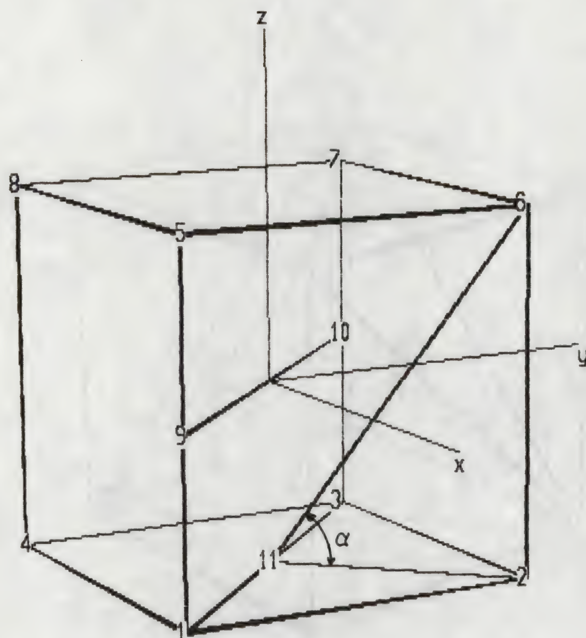
het assenstelsel, die tegelijk het centrum van het viervlak is; dit centrum is daarom een vast punt van de rotatie, met andere woorden, na een rotatie om de as 9-10 ligt het centrum van het viervlak nog steeds in O. Dit maakt het ons bijvoorbeeld gemakkelijk om een translatie die daarna moet volgen te specificeren (wat nodig is om figuur 3.4 te verkrijgen). De hoek α waarover de rotatie moet plaatsvinden is aangegeven in figuur 3.6. Het is de ingesloten hoek in punt 11 van de rechthoekige driehoek 11-2-6. De tegenovergelegen zijde (ribbe 2-6 van de kubus) heeft de lengte 1, en zijn aanliggende zijde (een halve diagonaal van een grensvlak van een kubus) heeft de lengte $\frac{1}{2}\sqrt{2}$. We hebben dus

$$\tan \alpha = 1/\tfrac{1}{2}\sqrt{2} = \sqrt{2}.$$

Gebruik makend van een zakrekenmachine vinden we

$$\alpha = \arctan \sqrt{2} = 54.73561°$$

We hebben deze hoek nodig om de gewenste rotatie te kunnen uitvoeren; na het verwijderen van de punten 5, 7, 2, 4 verkrijgen we het viervlak getoond linksboven in figuur 3.4.

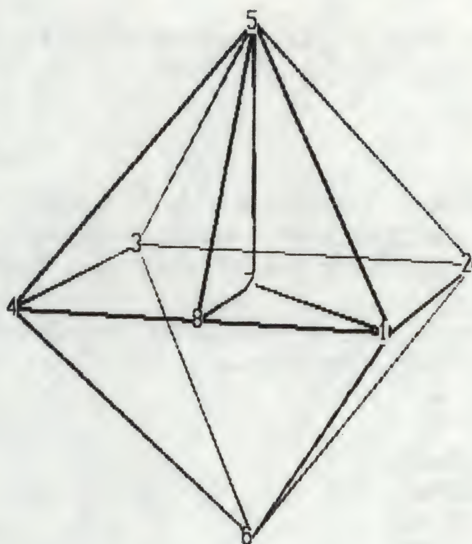


Figuur 3.6 Hoek α en de as 9-10, gebruikt in een rotatie

3.5.2 Zesvlak

Terwille van de uniformiteit in terminologie gebruiken we hier het woord *zesvlak* voor wat we gewoonlijk een *kubus* noemen. We weten natuurlijk al lang hoe we met D3D een kubus kunnen construeren, dus we hoeven dit niet nog eens in detail te bespreken.

Zoals u zich zult herinneren, kunnen we D3D gebruiken om twee voorwerpen afzonderlijk op te bouwen en ze dan één voor één van een file te lezen; we krijgen ze dan netjes naast elkaar door één ervan aan een translatie te onderwerpen, zoals we dat besproken hebben in paragraaf 2.2. Dit is de manier waarop we figuur 3.4 kunnen samenstellen: we maken de vijf voorwerpen één voor één en combineren ze pas als we vijf objectfiles hebben. In elk van deze vijf objectfiles bevindt het centrum van het voorwerp zich in de oorsprong O van het coördinatenstelsel. Op deze manier kunnen we O als het vaste punt in schalingsoperaties gebruiken, zodanig dat het voorwerp alleen maar groter of kleiner wordt, waarbij het centrum op zijn plaats blijft. We kunnen daarom het beste een eventuele schaling uitvoeren voordat we een translatie op het voorwerp in kwestie toepassen.



Figuur 3.7 Regelmatig achtvlak

3.5.3 Achtvlak

Zoals figuur 3.4 illustreert, heeft een (regelmatig) achtvlak twee hoekpunten die ter weerszijde van een vierkant gelegen zijn. Laten we puntnummers toekennen zoals is aangegeven in figuur 3.7 en laten we zeggen dat de zijden van vierkant 1-2-3-4 de lengte 1 hebben. Punt 7 is het centrum van het vierkant (en ook het centrum van het achtvlak) en punt 8 is het midden van de ribbe 4-1. Omdat punt 5 ligt op een loodlijn door punt 7, is de afstand *h* tussen deze twee punten het enige wat we nog moeten weten. We kunnen nu gebruik maken van het feit dat alle hoekpunten van een regelmatig veelvlak even ver van het centrum vandaan liggen. Zo is bijvoorbeeld driehoek 1-7-5 in figuur 3.7 een gelijkbenige rechthoekige driehoek: punt 7 is even ver van punt 5 als van punt 1 verwijderd en laatstbedoelde afstand is de lengte van de halve diagonaal van vierkant 1-2-3-4 (met zijden ter lengte 1). Dus als punt 7 de oorsprong van het coördinatenstelsel is en punt 5 ligt op de positieve z-as, dan zijn de z-coördinaten van de punten 5 en 6 gelijk aan $\frac{1}{2}\sqrt{2}$ en $-\frac{1}{2}\sqrt{2}$.

Het is goed om te controleren dat de grensvlakken van het aldus geconstrueerde lichaam werkelijk gelijkzijdige driehoeken zijn. We passen daarom eerst de stelling van Pythagoras toe op driehoek 8-7-5 om de lengte van lijnstuk 8-5 te vinden. Nadat we de lengten van de zijden 8-7 en 7-5 gekwadrateerd hebben berekenen we de som van die kwadraten:

$$(½)^2 + (½\sqrt{2})^2 = 3/4$$

Volgens Pythagoras is dit het kwadraat van de lengte van lijnstuk 8-5. We tellen de gekwadrateerde lengte van lijnstuk 8-1 erbij op en vinden

$$3/4 + (1/2)^2 = 1,$$

wat, alweer volgens Pythagoras, de lengte van de ribbe 1-5 moet zijn. Alle ribben van het lichaam hebben dus de lengte 1, wat betekent dat alle acht grensvlakken gelijkzijdige driehoeken zijn, zodat het lichaam inderdaad een regelmatig achtvlak is.
Omdat $½\sqrt{2} \approx 0.707107$, kan de volgende objectfile voor het achtvlak van figuur 3.7 (zonder de punten 7 en 8 erin) worden gebruikt:

```
1   0.5 0.5   0.000000
2  -0.5 0.5   0.000000
3  -0.5 0.5   0.000000
4   0.5 0.5   0.000000
5   0.0 0.0   0.707107
6   0.0 0.0   0.707107
Faces:
1   2   5.
2   3   5.
3   4   5.
4   1   5.
1   6   2.
2   6   3.
3   6   4.
4   6   1.
```

3.5.4 Twaalfvlak

Zoals vermeld aan het begin van paragraaf 3.5, heeft een regelmatig twaalfvlak (of dodecaëder) 12 grensvlakken, 30 ribben en 20 hoekpunten. Elk van de 12 grensvlakken is een regelmatige vijfhoek, dat wil zeggen een veelhoek met vijf gelijke zijden en vijf gelijke hoeken. Zoals we al in paragraaf 1.5 hebben gezien, past een twaalfvlak precies in een kubus; we kunnen van deze eigenschap gebruik maken als we een twaalfvlak willen construeren.

Voordat we regelmatige vijfhoeken gaan gebruiken in een twaalfvlak, willen we eerst eens zo'n vijfhoek zelf wat preciezer bekijken. In figuur 3.8 zien we er een,

waarvan het middelpunt E is; laten we zeggen dat de vijf hoekpunten van de vijfhoek liggen op een cirkel met straal 1 en middelpunt E.

Figuur 3.8 Regelmatige vijfhoek

Figuur 3.8 toont verschillende belangrijke hoeken. Hieruit en uit $EG = R = 1$, vinden we:

$$BE = \cos 36°$$
$$BG = \sin 36°$$

De hoek van 36° ($= \pi/5$ radialen) is bijzonder interessant, wat verband houdt met de volgende gelijkheden:

$$2 \times 36° = 72°$$
$$½ \times 36° = 18°$$
$$72° + 18° = 90°$$

Dit stelt ons in staat een eenvoudige uitdrukking voor cos 36° te vinden, namelijk

$$\cos 36° = (1 + \sqrt{5})/4$$

Zoals D. E. Knuth laat zien in *The Art of Computer Programming*, Vol. 1, Section 1.2.8, Exercise 19, kunnen we dit bewijzen door gebruik te maken van de variabelen u en v, gedefinieerd als:

$$u = \cos 72°, v = \cos 36°.$$

Er geldt dan:

$$u = 2 \cos^2 36° - 1 = 2v^2 - 1$$
$$v = 1 - 2 \sin^2 18° = 1 - 2 \cos^2 72° = 1 - 2u^2$$

Vandaar:

$$u + v = 2(v^2 - u^2) = 2(v + u)(v - u)$$

Delen door $u + v$ geeft:

$$1 = 2(v - u) = 2\{v - (2v^2 - 1)\} = -4v^2 + 2v + 2$$

We moeten dus de kwadratische vergelijking

$$4v^2 - 2v - 1 = 0,$$

oplossen, wat het gewenste resultaat geeft:

$$v = (1 + \sqrt{5})/4.$$

In werkelijkheid zullen we de nog belangrijker constante

$$\tau = (1 + \sqrt{5})/2 = 1.6180339887....$$

gebruiken, zodat we hebben:

$$\cos 36° = \tau/2$$

Het getal τ heeft de mooie eigenschap dat het kwadrateren ervan hetzelfde oplevert als het verhogen ervan met 1:

$$\tau^2 = (1 + \sqrt{5})^2/4 = 1\tfrac{1}{2} + \tfrac{1}{2}\sqrt{5}$$

dus we hebben:

$$\tau^2 = \tau + 1 \qquad (3.1)$$

Vermenigvuldigen we beide leden van deze vergelijking met machten van τ dan vinden we:

$$\tau^3 = \tau^2 + \tau$$
$$\tau^4 = \tau^3 + \tau^2$$
$$\tau^5 = \tau^4 + \tau^3$$
$$\dots$$

Evenzo, als we vergelijking (3.1) delen door machten van τ, dan krijgen we:

$$\tau = 1 + \tau^{-1}$$
$$1 = \tau^{-1} + \tau^{-2}$$
$$\tau^{-1} = \tau^{-2} + \tau^{-3}$$
$$\tau^{-2} = \tau^{-3} + \tau^{-4}$$
$$\dots$$

Dus elk element van

$$\dots, \tau^{-3}, \tau^{-2}, \tau^{-1}, 1, \tau, \tau^2, \tau^3, \dots$$

is de som van zijn onmiddellijke twee voorgangers.

Uit wat we gevonden hebben volgt dat als we een lijnstuk in twee stukken verdelen zodanig dat het langste stuk τ keer zo lang als het kortste stuk is, dat dan ook het gehele lijnstuk τ keer zo lang als het langste stuk is. Dit principe staat bekend als de *gulden snede*.

Een andere belangrijke eigenschap van τ is dat elke macht ervan geschreven kan worden als $a\tau + b$, waarin a en b gehele getallen zijn:

$$\tau^2 = \tau + 1$$
$$\tau^3 = \tau^2 + \tau = 2\tau + 1$$
$$\tau^4 = \tau^3 + \tau^2 = 3\tau + 2$$
$$\tau^5 = \tau^4 + \tau^3 = 5\tau + 3$$
$$\tau^6 = \tau^5 + \tau^4 = 8\tau + 5$$

enzovoort. Ook hebben we:

$$\tau^{-1} = \tau - 1$$
$$\tau^{-2} = 1 - \tau^{-1} = 2 - \tau$$
$$\tau^{-3} = \tau^{-1} - \tau^{-2} = 2\tau - 3$$
$$\tau^{-4} = \tau^{-2} - \tau^{-3} = 5 - 3\tau$$
$$\tau^{-5} = \tau^{-3} - \tau^{-4} = 5\tau - 8$$ enzovoort.

Gebruik makend van vergelijking (3.1) kunnen we nu gemakkelijk de identiteit

$$(a\tau + b)(a\tau - b - a) = a^2 - ab - b^2,$$

verifiëren, die (als a en b niet beide nul zijn) het volgende impliceert:

$$\frac{1}{a\tau + b} = \frac{a\tau - b - a}{a^2 - ab - b^2}$$

Dus elk quotiënt $P(\tau)/Q(\tau)$, waarin $P(\tau)$ en $Q(\tau)$ veeltermen in τ met rationale coëfficiënten zijn, kan worden geschreven als een lineaire vorm $a\tau + b$, waarin a en b rationale getallen zijn. Dit maakt τ een prettige constante om mee te werken, want als hij in een ingewikkelde uitdrukking voorkomt dan is er een goede kans dat we die uitdrukking kunnen vereenvoudigen.

Het lijkt misschien dat we afdwalen van ons onderwerp (de regelmatige vijfhoek) maar toch is dat niet zo. Om een enkel punt te noemen, elk tweetal diagonalen van een regelmatige vijfhoek die elkaar in een inwendig punt snijden verdelen elkaar volgens de gulden snede! We zullen hiervan geen gebruik maken, maar als we, zoals figuur 3.8 laat zien, enkele hulplijnen in een regelmatige vijfhoek aanbrengen, dan ontstaan er verscheidene hoeken van 36°, 72°, 18° en 54° (= 36° + 18°). We kunnen nu schrijven:

$$c = \cos 36° = \tau/2$$
$$s = \sin 36° = \surd(1 - c^2) = \surd(1 - \tau^2/4) = \surd\{1 - (\tau + 1)/4\}$$
$$= \tfrac{1}{2}\surd(3 - \tau)$$
$$\cos 72° = 2c^2 - 1 = \tfrac{1}{2}\tau^2 - 1 = \tfrac{1}{2}(\tau + 1) - 1 = (\tau - 1)/2$$

en, gebruik makend van de letters A t/m F als aangegeven in figuur 3.8, hebben we:

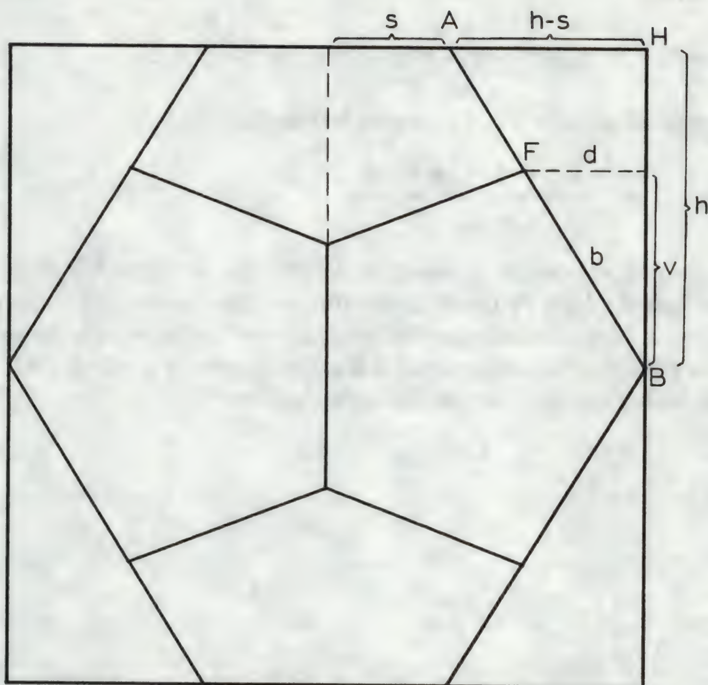
$$AB = AE + EB = 1 + \cos 36° = 1 + \tau/2$$
$$EF = ED \,.\, \cos 72° = (\tau - 1)/2$$

$$b = BF = BE + EF = \cos 36° + \cos 72° = \tau/2 + (\tau - 1)/2$$
$$= \tau - \tfrac{1}{2}$$

$$a = CF = FD = ED \,.\, \sin 72° = 1 \,.\, 2 \,.\, \cos 36° \,.\, \sin 36° = \tau s.$$

(We houden in gedachten dat we s in uitdrukkingen zoals τs kunnen vervangen door $\tfrac{1}{2}\surd(3 - \tau)$.)

We bekijken nu figuur 3.9, dat een vooraanzicht van een twaalfvlak in een kubus voorstelt, zoals we dat in figuur 1.12 in perspectief hebben gezien.



Figuur 3.9 Twaalfvlak in kubus (vooraanzicht)

Hier verschijnt AB (ook af te lezen in Fig 3.8) op ware lengte $1 + \tau/2$. Dit lijnstuk is de hypotenusa van de rechthoekige driehoek ABH, die we kunnen gebruiken om de halve ribbe h van de kubus te berekenen. Gebruik makend van Pythagoras, hebben we:

$$
\begin{aligned}
&AH^2 + BH^2 = AB^2 \\
&(h - s)^2 + h^2 = (1 + \tau/2)^2 \\
&2h^2 - 2sh + s^2 = (1 + \tau/2)^2 \\
&2h^2 - 2sh + (3 - \tau)/4 = 1 + \tau + \tau^2/4 \\
&2h^2 - 2sh + (3 - \tau)/4 = 1 + \tau + (\tau + 1)/4 \\
&4h^2 - 4sh - 3\tau - 1 = 0 \\
&h = [4s + \surd\{16s^2 + 16(3\tau + 1)\}]/8 \\
&= [4s + \surd\{4(3 - \tau) + 16(3\tau + 1)\}]/8, \\
&= \{2s + \surd(11\tau + 7)\}/4,
\end{aligned}
$$

wat we nog kunnen vereenvoudigen door gebruik te maken van

$$
\begin{aligned}
&\tau^4 = 3\tau + 2 \\
&\tau^6 = 8\tau + 5
\end{aligned}
$$

We vinden dan

$$\sqrt{(11\tau + 7)} = \sqrt{(\tau^6 + \tau^4)} = \tau^2\sqrt{(\tau^2 + 1)} = (\tau + 1)\sqrt{(\tau + 2)},$$

zodat we de volgende uitdrukking voor *h* krijgen:

$$h = \frac{2s + (\tau + 1)\sqrt{(\tau + 2)}}{4}$$

Met dit resultaat kennen we zowel de grootte van de kubus als de positie van de punten A en B. We willen ook de driedimensionale positie van de punten C en D weten, die in figuur 3.8 aangegeven zijn. Deze punten liggen op een horizontale lijn door het punt *f* dat zowel in figuur 3.8 als in figuur 3.9 te zien is, dus we bepalen eerst de positie van dit punt F. In figuur 3.9 hebben we

$$\frac{v}{b} = \frac{h}{\tau/2 + 1}$$

Dus

$$v = \frac{2bh}{\tau + 2} = \frac{2(\tau - ½)h}{\tau + 2} = (\tau - 1)h.$$

(De laatste gelijkheid kan worden geverifieerd door het produkt van $\tau + 2$ en $\tau - 1$ te berekenen en vergelijking (3.1) te gebruiken.)

Gebruik makend van deze uitdrukking voor *v* en van

$$\frac{d}{v} = \frac{h - s}{h}$$

(zie figuur 3.9), krijgen we

$$d = (\tau - 1)(h - s)$$

De posities van C en D (zie figuur 3.8) zijn nu gemakkelijk te vinden, aangezien zij liggen op de lijn door F die loodrecht staat op het vlak van figuur 3.9 en aangezien we de lengte van CF en DF hierboven gevonden hebben als $a = \tau s$.

Dankzij de symmetrie volgen alle andere hoekpunten van het twaalfvlak uit de hoekpunten die we hebben besproken. Programma DODECA laat dit in detail zien; het genereert een objectfile voor zowel het twaalfvlak als de kubus, zoals figuur 3.10 deze laat zien. Om de figuur niet nodeloos ingewikkeld te maken zijn de

coördinaatassen weggelaten, maar zij hebben de gebruikelijke richtingen, dat wil zeggen, zij hebben respectievelijk de richtingen van de ribben 103-102, 101-102 en 102-106 en de oorsprong ligt in het centrum van de kubus. Hoekpunt 2 komt overeen met punt A in de figuren 3.8 en 3.9 enzovoort. Als we alleen het twaalfvlak willen zien, dan kunnen gemakkelijk de kubus verwijderen door het *Delete*-commando toe te passen op het puntnummerbereik 101-108. Deze wenselijkheid doet zich bijvoorbeeld voor als we de geproduceerde objectfile met de andere regelmatige veelvlakken willen combineren, om aldus figuur 3.4 samen te stellen.

```
/* DODECA: Dit programma construeert een regelmatig twaalfvlak
           en een kubus waarin hij precies past.
*/
#include <stdio.h>
#include <math.h>

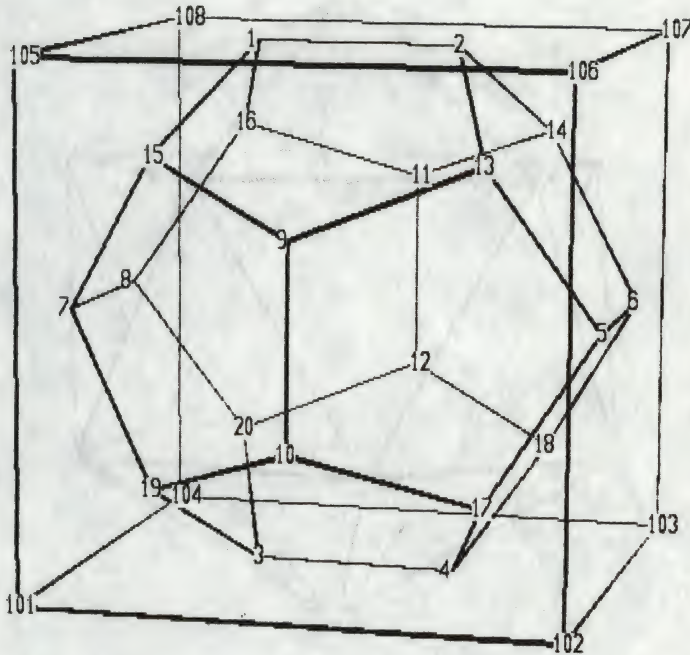
main()
{ float s, a, b, h, v, d, tau;
  static float x[21], y[21], z[21]; /* Beginwaarde 0 */
  int i;
  FILE *fp;
  tau = (sqrt(5.0) + 1)/2;          /*  tau = 2 * cos(pi/5)  */
  /* Soms wordt de letter phi in plaats van tau gebruikt. */
  s = sqrt(3 - tau)/2 ;             /* s = sin (pi/5)      */
  a = tau * s;
  h = (2*s+(tau+1)*sqrt(tau+2))/4;    /* Halve kubusribbe */
  b = tau - 0.5;
  v = (tau - 1) * h;
  d = (tau - 1) * (h - s);
  printf("De grensvlakken van het twaalfvlak zijn\n");
  printf("regelmatige vijfhoeken, die elk passen in\n");
  printf("een cirkel met straal 1.\n");
  printf("De zijden van deze vijfhoeken hebben de lengte %f.\n",
         2*s);
  printf(
  "Het twaalfvlak past in een kubus met ribben ter lengte");
  printf(" %f.\n", 2*h);
  x[5] = x[7] = s;
  x[6] = x[8] = -s;
  x[9] = x[10] = h;
  x[11] = x[12] = -h;
  x[13] = x[15] = x[17] = x[19] = a;
  x[14] = x[16] = x[18] = x[20] = -a;
  y[1] = y[3] = -s;
  y[2] = y[4] = s;
```

```
y[5] = y[6] = h;
y[7] = y[8] = -h;
y[13] = y[14] = y[17] = y[18] = h-d;
y[15] = y[16] = y[19] = y[20] = -(h-d);
z[1] = z[2] = h;
z[3] = z[4] = -h;
z[9] = z[11] = s;
z[10] = z[12] = -s;
z[13] = z[14] = z[15] = z[16] = v;
z[17] = z[18] = z[19] = z[20] = -v;
fp = fopen("dodeca.dat", "w");
for (i=1; i<=20; i++)
  fprintf(fp, "%3d %f %f %f\n", i, x[i], y[i], z[i]);

/* De kubus waarin de twaalfhoek past zal de
   hoekpuntnummers 101, 102, ..., 108 hebben:
*/
fprintf(fp, "101 %f %f %f\n",  h, -h, -h);
fprintf(fp, "102 %f %f %f\n",  h,  h, -h);
fprintf(fp, "103 %f %f %f\n", -h,  h, -h);
fprintf(fp, "104 %f %f %f\n", -h, -h, -h);
fprintf(fp, "105 %f %f %f\n",  h, -h,  h);
fprintf(fp, "106 %f %f %f\n",  h,  h,  h);
fprintf(fp, "107 %f %f %f\n", -h,  h,  h);
fprintf(fp, "108 %f %f %f\n", -h, -h,  h);

fprintf(fp, "Faces:\n");
fprintf(fp, " 1 15  9 13  2.\n");
fprintf(fp, " 1  2 14 11 16.\n");
fprintf(fp, " 5  6 14  2 13.\n");
fprintf(fp, " 7 15  1 16  8.\n");
fprintf(fp, "19 10  9 15  7.\n");
fprintf(fp, "10 17  5 13  9.\n");
fprintf(fp, "20  8 16 11 12.\n");
fprintf(fp, "18 12 11 14  6.\n");
fprintf(fp, " 3  4 17 10 19.\n");
fprintf(fp, " 4 18  6  5 17.\n");
fprintf(fp, " 3 20 12 18  4.\n");
fprintf(fp, " 3 19  7  8 20.\n");
/* De ribben van de kubus worden opgegeven als
   losse lijnstukken:
*/
fprintf(fp, "101 102.\n102 103.\n103 104.\n104 101.\n");
fprintf(fp, "105 106.\n106 107.\n107 108.\n108 105.\n");
```

```
    fprintf(fp, "101 105.\n102 106.\n103 107.\n104 108.\n");
    fclose(fp);
}
```



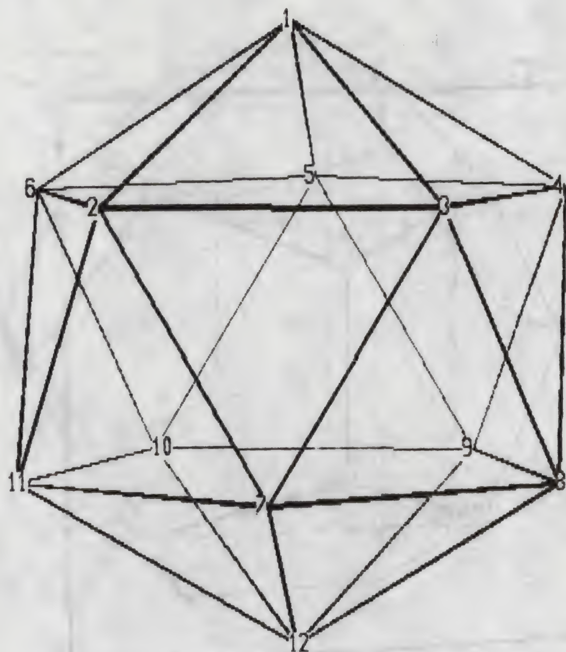
Figuur 3.10 Puntnummering in twaalfvlak met kubus

3.5.5 Twintigvlak

Zoals figuur 3.11 laat zien, kunnen we een regelmatig *twintigvlak* (of icosaëder) zodanig opstellen dat tien hoekpunten ervan tevens de hoekpunten zijn van twee horizontaal gelegen regelmatige vijfhoeken. Laten we de twee overige hoekpunten de *noordpool* (bovenaan) en de *zuidpool* (onderaan) noemen en laten we de term *as* gebruiken voor de lijn door deze polen. Het centrum O van het twintigvlak zal de oorsprong van ons coördinatenstelsel zijn en de as van het twintigvlak zal samenvallen met de z-as. De twee zojuist genoemde vijfhoeken zullen dezelfde grootte hebben als de grensvlakken van het in paragraaf 3.5.4 besproken twaalfvlak. Dus voor elke vijfhoek is de straal van de omgeschreven cirkel gelijk aan 1 en hebben de zijden de lengte $2s$, waarin

$$s = \sin 36°$$

Elke zijde van onze vijfhoeken is tevens een ribbe van het twintigvlak, dus ook deze ribben hebben de lengte $2s$.



Figuur 3.11 Puntnummering in twintigvlak

Zoals figuur 3.11 laat zien, kan de onderste horizontale vijfhoek uit de bovenste worden afgeleid door deze een rotatie over 36° en een translatie in verticale zin te laten ondergaan. Dit betekent dat we alleen nog de coördinaten van de twee vijfhoeken en van de twee polen moeten zien te vinden. In figuur 3.12 is de bovenste horizontale vijfhoek in perspectief afgebeeld; in plaats van de puntnummers 2, 3, 4, 5, 6 gebruiken we nu de letters A, B, C, D, E.

Punt F is het centrum van vijfhoek ABCDE, O is het centrum van het hele twintigvlak, N is de noordpool en M is het midden van CN. We willen nu h en r bepalen, aangegeven in figuur 3.12.

Daar NFC een rechthoekige driehoek is kunnen we de stelling van Pythagoras toepassen om $h = \text{FN}$, uitgedrukt in s (= sin 36°) te vinden:

Figuur 3.12 Regelmatige vijfhoek en noordpool van twintigvlak

$$FN^2 = CN^2 - CF^2$$
$$h^2 = (2s)^2 - 1^2$$
$$h = \sqrt{(4s^2 - 1)}$$

Alweer gebruik makend van $\tau = (1 + \sqrt{5})/2$ (zie paragraaf 3.5.4) en van

$$s = \tfrac{1}{2}\sqrt{(3 - \tau)}$$
$$s^2 = (3 - \tau)/4,$$

kunnen we h ook in τ uitdrukken:

$$h = \sqrt{(2 - \tau)} = \sqrt{(\tau^{-2})} = \tau^{-1} = \tau - 1$$

Teneinde $r = ON$ te vinden, merken we op dat driehoek NMO gelijkvormig is met driehoek NFC. We hebben daarom

$$\frac{s}{r} = \frac{h}{2s}$$

zodat

$$r = \frac{2s^2}{h} = \frac{(3 - \tau)/2}{\tau - 1} = \tau - \tfrac{1}{2}$$

De z-coördinaat van alle punten in vijfhoek ABCDE is gelijk aan

$$r - h = (\tau - ½) - (\tau - 1) = ½$$

Dit betekent dat de twee horizontale vijfhoeken op een afstand 1 van elkaar af liggen, met andere woorden hun onderlinge afstand is gelijk aan de straal van de omgeschreven cirkel van de vijfhoeken.

We hebben nu gevonden dat de hoekpunten 2, 3, 4, 5, 6 (zie figuur 3.11) de z-coördinaat 0.5 (= r - h) hebben en de hoekpunten 7, 8, 9, 10, 11 de z-coördinaat -0.5. De twee polen, 1 en 12, hebben respectievelijk de zcoördinaten $r = \tau$ - ½ en $-r = -(\tau$ - ½) en voor deze twee hoekpunten geldt $x = y = 0$. Wat de x- en y-coördinaten van de hoekpunten 2, ..., 11 betreft, gebruiken we het feit dat zij liggen op cirkels met straal 1. Voor geschikte hoeken α zijn die coördinaten respectievelijk gelijk aan cos α en sin α. We zullen deze hoek α als volgt kiezen:

Hoekpunt	α
2	-36°
3	36°
4	108°
5	180°
6	252°
7	0°
8	72°
9	144°
10	216°
11	288°

Van het bovenstaande wordt gebruik gemaakt in programma ICOSA, dat hieronder is opgenomen. Het genereert een objectfile die programma D3D in staat stelt figuur 3.11 te produceren.

```
/* ICOSA: Regelmatig twintigvlak (icosaeder); elk van de
          20 grensvlakken van dit lichaam is een
          gelijkzijdige driehoek.
*/
#include <stdio.h>
#include <math.h>
#define p(i, x, y, z) fprintf(fp,"%d %f %f %f\n",i,x,y,z)
#define face(i, j, k) fprintf(fp, "%d %d %d.\n", i, j, k)
```

```
main()
{ FILE *fp;
  int i;
  double s, pi, r, alpha, tau;
  pi = 4 * atan(1.0);
  tau = (sqrt(5.0) + 1)/2; /* tau = 2 * cos(pi/5);       */
  s = sin(pi/5);           /* pi/5 radialen = 36 graden */
  r = tau - 0.5;
  printf("Tien van de twaalf hoekpunten van het twintigvlak\n");
  printf(
"liggen op twee horizontale vijfhoeken, waarvan elk\n");
  printf("past in een circle met straal 1.\n");
  printf("Deze twee vijfhoeken liggen een afstand 1 van elkaar.\n");
  printf("Hun zijden hebben de lengte %f.\n", 2*s);
  printf(
"Het twintigvlak past in een bol met straal %f.\n", r);
  fp = fopen("icosa.dat", "w");
  p(1, 0.0, 0.0, r); /* Noordpool */
  for (i=0; i<5; i++)
  { alpha = -pi/5 + i * pi/2.5;
                         /* In graden: -36 + i * 72 */
    p(2+i, cos(alpha), sin(alpha), 0.5);
  }
  for (i=0; i<5; i++)
  { alpha = i * pi/2.5;
    p(7+i, cos(alpha), sin(alpha), -0.5);
  }
  p(12, 0.0, 0.0, -r);  /* Zuidpool */
  fprintf(fp, "Faces:\n");
  for (i=0; i<5; i++) face(1, 2+i, i<4 ? 3+i : 2);
  for (i=0; i<5; i++) face(2+i, 7+i, i<4 ? 3+i : 2);
  for (i=0; i<5; i++)
              face(7+i, i<4 ? 8+i : 7, i<4 ? i+3 : 2);
  for (i=0; i<5; i++) face(i+7, 12, i<4 ? i+8 : 7);
}
```

Het is misschien aardig te vermelden dat er een andere manier is om een regelmatig twintigvlak te construeren. Deze elegante methode is gebaseerd op drie onderling loodrechte rechthoeken, vergelijkbaar met de drie onderdelen die het voorwerp vormen dat we zagen in figuur 2.22. In figuur 3.11 kunnen we bijvoorbeeld de rechthoeken

2-6-9-8,
1-4-12-11,
3-7-10-5

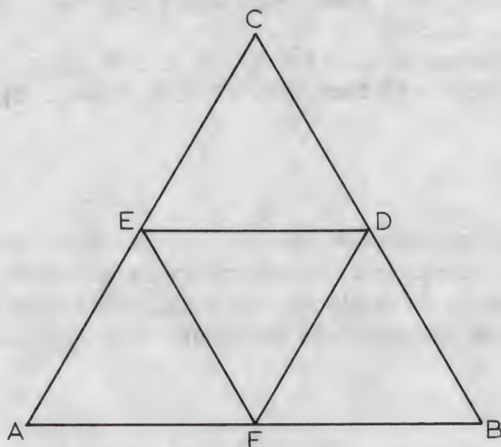
voor dit doel gebruiken. Dit zijn zogeheten *gulden* rechthoeken: hun zijden zijn in de verhouding 1 : τ, dat wil zeggen de breedte staat tot de lengte als de lengte staat tot de som van lengte en breedte. (We krijgen dus de lengte en breedte door op die som de 'gulden snede' toe te passen, vandaar de naam 'gulden rechthoek'.) Ik zou u willen aanraden deze methode bij wijze van oefening eens toe te passen en op die manier een alternatief programma voor het genereren van een regelmatig twintigvlak te schrijven.

3.6 EEN BOL BENADERD DOOR EEN TACHTIGVLAK

Alle vijf de regelmatige veelvlakken hebben te weinig grensvlakken om gebruikt te kunnen worden als benadering van een bol. We kunnen evenwel een regelmatig twintigvlak gebruiken als basis voor een ander lichaam, dat 80 driehoekige grensvlakken heeft. We hebben dit lichaam in het kort besproken in paragraaf 2.2; het is afgebeeld in de figuren 2.9 en 2.10.

Zoals figuur 3.13 laat zien, kunnen we een driehoek verdelen in vier kleinere driehoeken door het midden van elke zijde te gebruiken en naburige punten te verbinden. Letterlijke toepassing hiervan op de 20 gelijkzijdige driehoeken van een twintigvlak zou geen zin hebben, want dan zouden we het aantal vlakken waarin alle grensvlakken liggen niet echt vergroten.

In plaats van de middens D, E en F (zie figuur 3.13) zelf te gebruiken, zullen we hun projecties nemen op het boloppervlak dat we willen benaderen. Laat deze bol de straal 1 en middelpunt O (de oorsprong van het coördinatenstelsel) hebben. De hoekpunten A, B en C van de grote driehoeken liggen dan op een afstand 1 van O, maar de punten D, E en F liggen op een iets kleinere afstand, zeg *d*, van O. We gaan



Figuur 3.13 Driehoek verdeeld in vier kleinere driehoeken

nu de lijnstukken OD, OE en OF (die de lengte d hebben) verlengen, waarbij de lijnstukken OD', OE' en OF' ontstaan, zodanig dat deze laatste drie lijnstukken de lengte 1 hebben. Zo berekenen we de coördinaten van bijvoorbeeld punt D' als volgt:

$$x_{D'} = x_D/d$$
$$y_{D'} = y_D/d$$
$$z_{D'} = z_D/d$$

In plaats van de middens D, E en F gaan we nu de punten D', E' en F' gebruiken en we vervangen zo de oorspronkelijke driehoek ABC door de vier kleinere driehoeken AF'E', F'BD', E'D'C en F'D'E'.

Programma SPH80 is gebaseerd op programma ICOSA van paragraaf 3.5.5. Ook nu berekenen we eerst de 12 hoekpunten van het twintigvlak, maar deze keer delen we de coördinaten door een factor r, om zodoende een bol met straal 1 (in plaats van een bol met straal r) te benaderen. Daar elke zijde van een grote driehoek ook optreedt als zijde van een aangrenzende driehoek, zouden we ieder punt in het midden van een zijde twee keer genereren als we geen bijzondere maatregelen namen. We houden daarom bij van welke ribben het midden al berekend is en zorgen ervoor dat elk punt maar één keer in de objectfile voorkomt. De programmatekst zelf geeft verdere bijzonderheden.

```
/* SPH80: Tachtigvlak voor het benaderen van een bol.
      Dit lichaam is gebaseerd op een regelmatig twintig-
      vlak dat 20 gelijkzijdige driehoeken als grensvlakken
      heeft. In plaats van elk van deze driehoeken worden
      vier kleinere driehoeken gebruikt, hetgeen 20 x 4 = 80
      van deze kleinere driehoeken geeft. (Deze zijn niet
      gelijkzijdig.) Voor elke gelijkzijdige driehoek van het
      twintigvlak worden de middens van de drie zijden gepro-
      jecteerd op de bol waarin het twintigvlak past. We ver-
      krijgen de vier kleinere driehoeken door naburige punten
      met elkaar te verbinden.
*/
#include <stdio.h>
#include <math.h>
#include <process.h>

void point(int i, float x, float y, float z);
void subdivide(int A, int B, int C);
void storeface(int i, int j, int k);
void midpnt(int B, int C, int *pP, float x, float y,
                                            float z);
```

```
int npoints;
struct {float x, y, z;} pnt[43];

int nface=0;
struct {int i, j, k;} fc[80];

int nedge=0;
struct {int i, j, P;} edges[120];

FILE *fp;

main()
{ int i, l;
  double pi, r, alpha, tau;
  pi = 4 * atan(1.0);
  tau = (sqrt(5.0) + 1)/2;  /* tau = 2 * cos(pi/5)  */
  r = tau - 0.5;
  printf("Een bol (met straal 1) wordt benaderd door een\n");
  printf("lichaam met 80 driehoekige grensvlakken. Het\n");
  printf("resultaat wordt geschreven naar de objectfile\n");
  printf("SPH80.DAT (die door D3D gelezen kan worden).\n");
  fp = fopen("SPH80.DAT", "w");
  point(1, 0.0, 0.0, 1.0); /* Noordpool */
  for (i=0; i<5; i++)
  { alpha = -pi/5 + i * pi/2.5;
                /* In graden: -36 + i * 72 */
    point(2+i, cos(alpha)/r, sin(alpha)/r, 0.5/r);
  }
  for (i=0; i<5; i++)
  { alpha = i * pi/2.5;
    point(7+i, cos(alpha)/r, sin(alpha)/r, -0.5/r);
  }
  point(12, 0.0, 0.0, -1.0); /* Zuidpool */
  npoints = 12;
  for (i=0; i<5; i++) subdivide(1, 2+i, i<4 ? 3+i : 2);
  for (i=0; i<5; i++) subdivide(2+i, 7+i, i<4 ? 3+i : 2);
  for (i=0; i<5; i++)
            subdivide(7+i, i<4 ? 8+i : 7, i<4 ? i+3 : 2);
  for (i=0; i<5; i++) subdivide(i+7, 12, i<4 ? i+8 : 7);
  fprintf(fp, "Faces:\n");
  for (l=0; l<80; l++)
    fprintf(fp, "%d %d %d.\n", fc[l].i, fc[l].j, fc[l].k);
  fclose(fp);
}
void midpnt(int B, int C, int *pP, float x, float y,
                                              float z)
```

```
/*  Punt (x, y, z) is het midden van BC.
    Als het een nieuw hoekpunt is, bewaar het dan en
    schrijf het naar de objectfile, gebruik makend van
    een nieuw puntnummer. Zo niet, vind het nummer van
    het punt. Het puntnummer moet in elk geval aan
    *pP worden toegekend.
*/
{ int tmp, e;
  if (C < B) {tmp = B; B = C; C = tmp;}
  /* B, C terwille van de eenduidigheid in opklimmende volgorde */
  for (e=0; e<nedge; e++)
    if (edges[e].i == B && edges[e].j == C) break;
  if (e == nedge)  /* Niet gevonden, dus een nieuw hoekpunt */
  { edges[e].i = B; edges[e].j = C;
    edges[e].P = *pP = ++npoints;
    nedge++;
    point(*pP, x, y, z);
  } else *pP = edges[e].P;
           /* Ribbe BC is al eerder behandeld, dus
              het punt is niet nieuw.
           */
}

void point(int i, float x, float y, float z)
{ fprintf(fp, "%d %f %f %f\n", i, x, y, z);
  pnt[i].x = x; pnt[i].y = y; pnt[i].z = z;
}

void subdivide(int A, int B, int C)
/* Verdeel driehoek ABC in vier kleinere driehoeken
*/
 { float xP, yP, zP, xQ, yQ, zQ, xR, yR, zR;
   int P, Q, R;
   static float d = -1;
     /* d is alleen gelijk aan -1 voordat deze functie
        de eerste keer wordt aangeroepen; daarna zal d
        steeds zijn correcte waarde (tussen 0 en 1) hebben,
        die gelijk is aan de afstand tussen een punt in het
        midden van een zijde en het middelpunt van de bol.
        We projecteren alle middens van zijden in een punt
        op de bol (met straal 1) door hun coordinaten
        door d te delen.
 */
xP = (pnt[B].x + pnt[C].x)/2;
yP = (pnt[B].y + pnt[C].y)/2;
zP = (pnt[B].z + pnt[C].z)/2;
```

```
    xQ = (pnt[C].x + pnt[A].x)/2;
    yQ = (pnt[C].y + pnt[A].y)/2;
    zQ = (pnt[C].z + pnt[A].z)/2;

    xR = (pnt[A].x + pnt[B].x)/2;
    yR = (pnt[A].y + pnt[B].y)/2;
    zR = (pnt[A].z + pnt[B].z)/2;

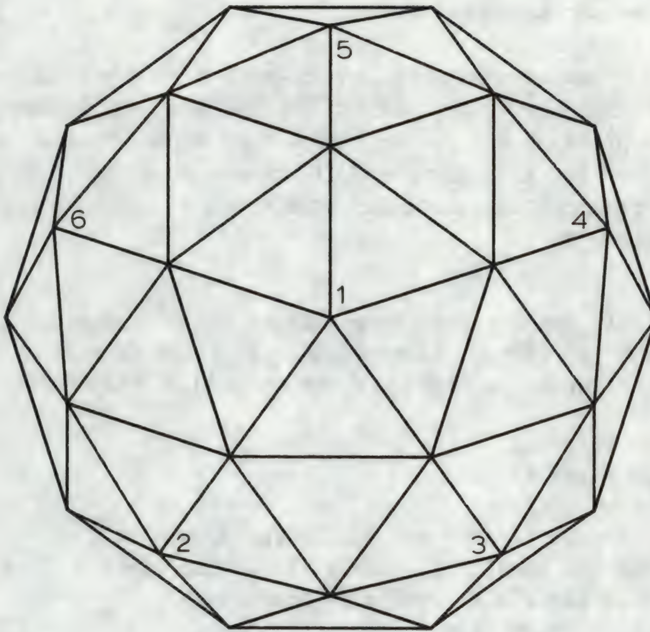
    if (d < 0) d = sqrt(xP*xP + yP*yP + zP*zP);
                                      /* 0 < d < 1 */
    xP /= d; yP /= d; zP /= d;
    xQ /= d; yQ /= d; zQ /= d;
    xR /= d; yR /= d; zR /= d;

    midpnt(B, C, &P, xP, yP, zP);
    midpnt(C, A, &Q, xQ, yQ, zQ);
    midpnt(A, B, &R, xR, yR, zR);

    storeface(A, R, Q);
    storeface(R, B, P);
    storeface(Q, P, C);
    storeface(Q, R, P);
  }

  void storeface(int i, int j, int k)
  { fc[nface].i = i;
    fc[nface].j = j;
    fc[nface].k = k;
    nface++;
  }
```

We hebben al verschillende perspectivische afbeeldingen van het geconstrueerde lichaam gezien (namelijk in de figuren 2.9 en 2.10). Figuur 3.14 toont een bovenaanzicht van het lichaam. Zoals bekend duidt punt 1 de noordpool aan; in dit aanzicht zien we de hoekpunten van de vijf oorspronkelijke driehoeken 1-2-3, 1-3-4, 1-4-5, 1-5-6, 1-6-2 en hoe elk hiervan vervangen is door vier kleinere driehoeken.
Het principe dat we zojuist hebben gebruikt kan nog eens op het nieuwe lichaam worden toegepast. Dit betekent dat het aantal driehoekige grensvlakken dan nog eens met 4 wordt vermenigvuldigd, zodat er 320 grensvlakken ontstaan. Het is een goede oefening dit verder uit te werken!



Figuur 3.14 Bovenaanzicht van tachtigvlak

3.7 DRIEDIMENSIONALE ROTATIES

In paragraaf 1.8.1 hebben we enige gebruikersaspecten besproken van rotaties in de driedimensionale ruimte. We gaan ons nu bezig houden met de software voor zulke rotaties. Van het gezichtspunt van de programmeur bekeken, worden rotaties uitgevoerd door middel van twee functies, *initrotate* en *rotate*. Met eerstgenoemde functie geven we gedetailleerde informatie over de rotatie op, zodat een *rotatiematrix* kan worden gegenereerd. Dit wordt voor elke rotatie maar één keer gedaan, ongeacht het aantal te roteren punten. De functie *rotate* doet het eigenlijke werk; hij wordt aangeroepen voor elk punt dat geroteerd moet worden.

Om een rotatie volledig te kunnen specificeren hebben we een gerichte as en een hoek α nodig. Voor de gerichte as gebruiken we een punt A, met coördinaten x_A, y_A, z_A, en een vector

$$\mathbf{v} = [v_1, v_2, v_3]$$

De gerichte as is dan de lijn door punt A met de richting van vector v. Als we een

(normale) schroef draaien op de manier van de bedoelde rotatie dan beweegt hij zich langs de as in de richting van vector v.

De functies *initrotate* en *rotate* zijn bijzonder gemakkelijk te gebruiken. Laten we bijvoorbeeld eens programma GENROTA bekijken; dit is een algemeen programma voor het uitvoeren van een willekeurige driedimensionale rotatie. Het programma leest een objectfile (in D3D-formaat) en schrijft een soortgelijke uitvoerfile. De namen van deze drie files, punt A, vector v en hoek α worden via het toetsenbord ingetypt:

```
/* GENROTA: Een algemeen programma om een verzameling
   punten te roteren. Na compilatie moet het door de
   linker worden samengevoegd met de module TRAFO.OBJ.
*/
#include <stdio.h>
#include <math.h>
#include <process.h>
void initrotate(double xA, double yA, double zA,
   double v1, double v2, double v3, double alpha);
void rotate(double x, double y, double z,
   double *px1, double *py1, double *z1);

main()
{ FILE *fp1, *fp2;
  double xA, yA, zA, v1, v2, v3, alphadeg, alpha,
         x, y, z, x1, y1, z1, pi;
  char str[30];
  int nr, ch, n=0;
  printf("Invoerfile:  "); scanf("%s", str);
  fp1 = fopen(str, "r");
  printf("Uitvoerfile: "); scanf("%s", str);
  fp2 = fopen(str, "w");
  if (fp1 == NULL || fp2 == NULL)
  { printf("File-probleem"); exit(1);
  }
  printf("Geef xA, yA, zA: ");
  scanf("%lf %lf %lf", &xA, &yA, &zA);
  printf("Geef v1, v2, v3: ");
  scanf("%lf %lf %lf", &v1, &v2, &v3);
  printf("Geef alpha (in graden): ");
  scanf("%lf", &alphadeg);
  pi = 4 * atan(1.0);
  alpha = alphadeg * pi/180; /* alpha in radialen */
  initrotate(xA, yA, zA, v1, v2, v3, alpha);
```

```
while (
fscanf(fp1, "%d %lf %lf %lf", &nr, &x, &y, &z) == 4)
{ rotate(x, y, z, &x1, &y1, &z1); n++;
  fprintf(fp2, "%d %f %f %f\n", nr, x1, y1, z1);
}

while (ch = getc(fp1), ch != EOF) putc(ch, fp2);
  fclose(fp1); fclose(fp2);
  printf("Klaar! %d punten geroteerd.\n", n);
}
```

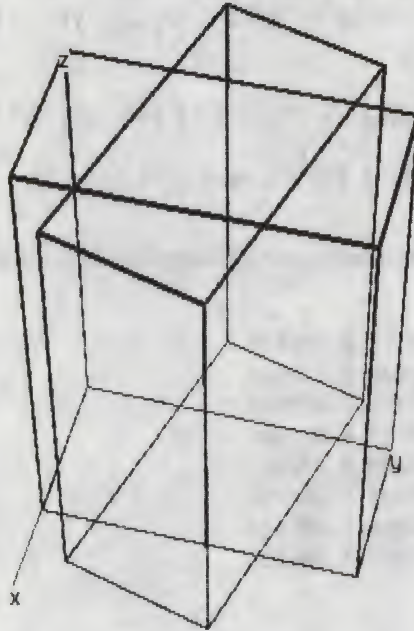
Laten we bij wijze van voorbeeld een prisma gebruiken, dat gegeven wordt door de volgende objectfile:

```
1 1.000000 0.000000 0.000000
2 1.000000 2.000000 0.000000
3 0.000000 2.000000 0.000000
4 0.000000 0.000000 0.000000
5 1.000000 0.000000 3.000000
6 1.000000 2.000000 3.000000
7 0.000000 2.000000 3.000000
8 0.000000 0.000000 3.000000
Faces:
1 2 6 5.
2 3 7 6.
3 4 8 7.
1 4 8 5.
5 6 7 8.
1 4 3 2.
```

Met $x_A = 0.5$, $y_A = 1$, $z_A = 0$, $v_1 = 0$, $v_2 = 0$, $v_3 = 1$, $\alpha = 75°$ wordt deze file getransformeerd in een nieuwe die hetzelfde prisma beschrijft, geroteerd over 75° om zijn verticale as. Zowel het oorspronkelijke als het geroteerde prisma is te zien in figuur 3.15.

Programma GENROTA moet na het compileren (door de zogenaamde *linker*) worden samengevoegd met de eveneens gecompileerde module TRAFO, die we nog zullen bespreken. Als u nog nooit met gebruik van Turbo C zoiets gedaan heeft, dan dient u te weten dat u gebruik kunt maken van een zogeheten *projectfile*, laten we zeggen GENROTA.PRJ, die in ons voorbeeld slechts twee regels tekst bevat, namelijk

```
GENROTA
TRAFO
```



Figuur 3.15 Prisma voor en na rotatie

U moet dan de Turbo-C-optie *P* kiezen en de naam van de projectfile opgeven en alles gaat verder zowat vanzelf. Wij zullen de module TRAFO voor verschillende toepassingen gaan gebruiken, dus het is een goed idee hem maar één keer te compileren. Als we deze module, die hieronder weergegeven is (en voluit geschreven TRAFO.C als filenaam heeft) compileren, dan ontstaat de objectfile TRAFO.OBJ. Als dit eenmaal gedaan is hoeft TRAFO daarna voor andere toepassingen niet meer te worden gecompileerd en evenmin hoeven we voor andere programma's, waarin we ook rotaties willen uitvoeren, de programmatekst van TRAFO te kopiëren.

```
/* TRAFO: Driedimensionale transformaties */

#include <math.h>

double r11, r12, r13, r21, r22, r23,
  r31, r32, r33, r41, r42, r43;
```

```
void initrotate(double a1, double a2, double a3,
          double v1, double v2, double v3, double alpha)
/* Berekening van de rotatiematrix

            | r11  r12  r13  0 |
        R = | r21  r22  r23  0 |
            | r31  r32  r33  0 |
            | r41  r42  r43  1 |

   de gebruikt wordt in

      [x1  y1  z1  1] = [x  y  z  1] R,

   zie de funktie 'rotate'.
   Punt (x1, y1, z1) is het beeld van (x, y, z).
   De rotatie vindt plaats om de as

     (a1, a2, a3)+lambda(v1, v2, v3)

   en over de hoek alpha.
*/
{ double rho, theta, cal, sal, cph, sph, cth, sth,
     cph2, sph2, cth2, sth2, pi, cal1;
  cal = cos(alpha); sal = sin(alpha);
  cal1 = 1.0-cal;
  rho = sqrt(v1*v1+v2*v2+v3*v3);
  pi = 4.0 * atan(1.0);
  if (rho == 0.0) {theta=0.0; cph=1.0; sph=0.0;} else
  { if (v1 == 0.0)
       theta = (v2 >= 0.0 ? 0.5*pi : 1.5*pi);
    else
    { theta = atan(v2/v1);
      if (v1 < 0) theta += pi;
    }
    cph = v3/rho; sph = sqrt(1.0 - cph*cph);
    /* cph = cos(phi), sph = sin(phi)  */
  }
  cth = cos(theta); sth = sin(theta);
  cph2 = cph*cph; sph2 = 1.0 - cph2;
  cth2 = cth*cth; sth2 = 1.0 - cth2;
  r11 = (cal*cph2+sph2)*cth2+cal*sth2;
  r12 = sal*cph+cal1*sph2*cth*sth;
  r13 = sph*(cph*cth*cal1-sal*sth);
  r21 = sph2*cth*sth*cal1-sal*cph;
```

```
    r22 = sth2*(cal*cph2+sph2)+cal*cth2;
    r23 = sph*(cph*sth*cal1+sal*cth);
    r31 = sph*(cph*cth*cal1+sal*sth);
    r32 = sph*(cph*sth*cal1-sal*cth);
    r33 = cal*sph2+cph2;
    r41 = a1-a1*r11-a2*r21-a3*r31;
    r42 = a2-a1*r12-a2*r22-a3*r32;
    r43 = a3-a1*r13-a2*r23-a3*r33;
  }

void rotate(double x, double y, double z,
       double *px1, double *py1, double *pz1)
{ *px1 = x*r11+y*r21+z*r31+r41;
  *py1 = x*r12+y*r22+z*r32+r42;
  *pz1 = x*r13+y*r23+z*r33+r43;
}
```

Als u met matrixvermenigvuldiging bekend bent dan zal de volgende toelichting u helpen de functies *initrotate* en *rotate* te begrijpen. Laatstgenoemde functie berekent de beeldvector x' door de oorspronkelijke vector x (waarvan het eindpunt geroteerd moet worden) als volgt te vermenigvuldigen met de algemene rotatiematrix R_{GEN}:

$$\mathbf{x}' = \mathbf{x}\, R_{GEN}$$

waarin

$$\mathbf{x} = [x\ y\ z\ 1]$$
$$\mathbf{x}' = [x'\ y'\ z'\ 1]$$

$$R_{GEN} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ r_{41} & r_{42} & r_{43} & 1 \end{bmatrix}$$

Het is de taak van functie *initrotate* deze matrix te berekenen. We vinden hem als produkt van drie andere matrices:

$$R_{GEN} = T^{-1} R^{*} T$$

waarin T een *translatiematrix* is, die de coördinaten van het gegeven punt A bevat:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_A & y_A & z_A & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_A & -y_A & -z_A & 1 \end{bmatrix}$$

Voordat we de rotatie uitvoeren, verschuiven we het voorwerp naar O door middel van T^{1}, wat ook een translatiematrix is. Dan vindt de rotatie om een as door O (in de richting van v) over de hoek α plaats, waarvoor matrix R^* zorgt, en tenslotte wordt het geroteerde voorwerp teruggeschoven door middel van matrix T. We moeten nu nog matrix R^* bespreken. Deze 4 x 4 matrix wordt afgeleid van de 3 x 3 matrix R door eenvoudig een eenheidskolomvector en een eenheidsrijvector aan laatstgenoemde matrix toe te voegen; dus we kunnen symbolisch het volgende noteren:

$$R^* = \begin{bmatrix} . & . & . & 0 \\ . & . & . & 0 \\ . & . & . & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De negen punten hierin stellen de elementen van matrix R voor, die nu de nog ontbrekende schakel is. Hij wordt gevonden als produkt van vijf andere 3 x 3 matrices:

$$R = R_z^{-1} R_y^{-1} R_v R_y R_z$$

De middelste matrix, R_v, voert de eigenlijke rotatie over hoek α uit:

$$R_v = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zoals onmiddellijk uit deze matrix volgt, voert R_v een rotatie om de z-as uit, terwijl een rotatie om de gerichte as door O in de richting van v vereist is. Dit verklaart de aanwezigheid van de andere vier matrices; zij zorgen voor coördinatentransformaties, zoals we in het kort zullen bespreken.

Allereerst hebben we de bolcoördinaten van het eindpunt (v_1, v_2, v_3) van vector v

nodig (aannemende dat O het beginpunt van deze vector is). Vrijwel in het begin van dit boek, in figuur 1.2, wordt aangegeven wat de betekenis van de bolcoördinaten ρ, θ, φ is. Zij staan als volgt in verband met het eindpunt van vector **v**:

$$v_1 = \rho \sin \varphi \cos \theta$$
$$v_2 = \rho \sin \varphi \sin \theta$$
$$v_3 = \rho \cos \varphi$$

Helaas moeten we niet v_1, v_2, v_3 berekenen uit ρ, θ, φ maar is het net andersom. We hebben daarom de omgekeerde betrekkingen nodig, die ingewikkelder zijn in de zin dat er logische beslissingen aan te pas komen:

$$\rho = \sqrt{(v_1^2 + v_2^2 + v_3^2)}$$

Als $\rho = 0$ (wat overigens niet mag voorkomen), stellen we $\theta = \varphi = 0$. Zo niet, dan hebben we:

$$\theta = \begin{cases} \arctan(v_2/v_1) & \text{als } v_1 > 0 \\ \pi + \arctan(v_2/v_1) & \text{als } v_1 < 0 \\ \pi/2 & \text{als } v_1 = 0 \text{ en } v_2 \geq 0 \\ 3\pi/2 & \text{als } v_1 = 0 \text{ en } v_2 < 0 \end{cases}$$

$$\varphi = \arccos(v_3/\rho)$$

We kunnen nu overgaan op een ander coördinatenstelsel, zodanig dat **v**, de as van rotatie, ligt op de nieuwe positieve z-as. We beginnen met een rotatie van de x- en de y-as om de z-as over een hoek θ. Als we op deze manier een punt moesten roteren, dan zou de rotatiematrix als volgt luiden:

$$R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Omdat we niet een punt maar het assenstelsel transformeren, hebben we de inverse van deze matrix nodig, dat wil zeggen:

$$R_z^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(Het feit dat we de inverse bewerking nodig hebben kunt u vergelijken met een eenvoudig voorbeeld in de tweedimensionale ruimte: als we de y-as een afstand *d* naar rechts schuiven, dan veranderen de x-coördinaten van alle punten op dezelfde

manier als wanneer we in plaats daarvan die punten een afstand *d* naar links verplaatsen.)

In de volgende stap roteren we de (nieuwe) x- en z-as om de (nieuwe) y-as over een hoek φ, waarvoor we de volgende matrix nodig hebben:

$$R_y^{-1} = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix}$$

Dit geeft een nieuwe z-as, die de richting van vector **v** heeft, dus we kunnen nu bovengenoemde matrix R_v gebruiken. Hierna dient het coördinatenstelsel weer naar zijn oorspronkelijke stand teruggetransformeerd te worden. We gebruiken daartoe eerst de matrix

$$R_y = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix}$$

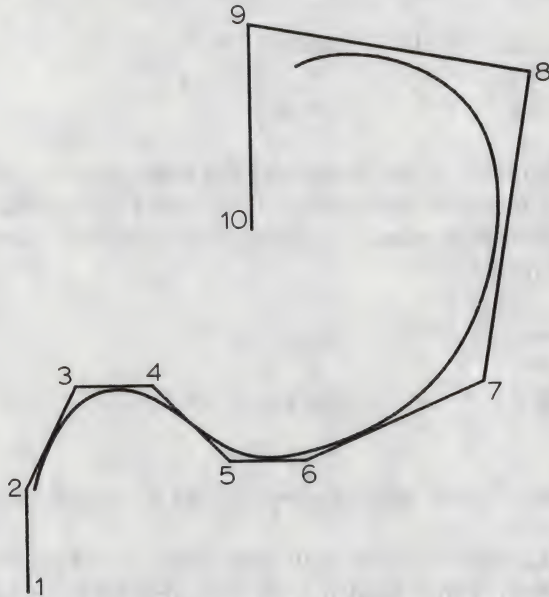
en daarna de matrix R_z, die we eerder, tegelijk met R_z^{-1}, hebben besproken.

In principe zou het berekenen van *R* als produkt van de vijf gegeven matrices door de computer gedaan kunnen worden, maar aangezien dat met getallen in plaats van met uitdrukkingen gedaan zou moeten worden zou dat een hoop onnodig rekenwerk met zich mee brengen. De uiteindelijke matrix *R* is namelijk eenvoudiger dan de tussentijdse resultaten en als we *R*, uitgedrukt in θ, φ en α, beschikbaar hebben, dan kunnen we deze onmiddellijk toepassen in plaats van iedere keer al die matrixvermenigvuldigingen uit te voeren. Ik heb daarom dit werk op de conventionele manier gedaan, gebruik makend van een groot vel papier en van allerlei afkortingen, zoals sα voor sin α, om de enorm lange formules die we tijdelijk nodig hebben op een redelijke manier te kunnen noteren. Gelukkig zijn hierbij een flink aantal vereenvoudigingen mogelijk. Het resultaat wordt gebruikt in de functie *initrotate*, die in de module TRAFO voorkomt.

3.8 B-SPLINE RUIMTELIJKE KROMMEN

In paragraaf 2.4 hebben we gebruik gemaakt van programma CURVE3 om een rij punten in de driedimensionale ruimte te benaderen door een kromme. We gaan nu dit programma zelf bespreken, tezamen met de B-spline approximatiemethode waarop het is gebaseerd. Het is sterk aan te bevelen paragraaf 2.4 nog eens over te lezen (zoals ik zelf voor het schrijven van de onderhavige paragraaf ook heb moeten doen!).

Bij m gegeven punten zijn er m - 1 intervallen, maar slechts m - 3 hiervan corresponderen met een stuk van de B-spline kromme. Laten we zo'n stuk kromme een *segment* noemen. In bijvoorbeeld figuur 3.16, waarin geldt $m = 10$, bestaat de hele kromme uit 10 - 3 = 7 segmenten, die corresponderen met de lijnstukken 2-3, 3-4, 4-5, 5-6, 6-7, 7-8 en 8-9.



Figuur 3.16 Punten, benaderd door een B-spline kromme

Een moeilijkheid bij B-spline approximatie is dat de hele kromme zo mooi vloeiend is dat we de segmenten waaruit hij is opgebouwd niet van elkaar kunnen onderscheiden! In figuur 3.16 zien we dat het eerste segment begint in de buurt van punt 2, dus het moet eindigen in de buurt van punt 3, waar het tweede segment begint, maar we kunnen niet zien waar precies de overgang plaatsvindt. Toch is het nodig ons te realiseren dat er zulke grenspunten zijn, want anders kunnen we de wiskundige functies waarop de methode gebaseerd is niet begrijpen.

Het is misschien aardig even te bespreken hoe figuur 3.16 tot stand is gekomen. Ik heb eerst D3D gebruikt om de punten 1 t/m 10 te definiëren en de negen verbindende rechte lijnstukken te tekenen. We zouden de figuur een 'vooraanzicht' kunnen noemen, dat wil zeggen het oogpunt ligt heel ver op de positieve x-as ($\rho = 100000$, $\theta = 0°$ en $\varphi = 90°$). De tien punten zijn met behulp van cursorbesturing gedefinieerd in het yz-vlak. Daarna heb ik programma CURVE3 op de objectfile, zeg PNT10.DAT, toegepast en $N = 20$ opgegeven als het aantal intervallen tussen elk tweetal opeenvolgende gegeven punten. Dit resulteerde in een uitvoerfile, die ik weer als invoer voor D3D heb gebruikt. Tenslotte heb ik, zonder het scherm eerst

schoon te maken, met commando *R* de oorspronkelijke file PNT10.DAT weer ingelezen om een figuur te verkrijgen waarin niet alleen de kromme maar ook de gegeven tien punten en de rechte lijnen ertussen te zien zijn. Wel moesten de tien puntnummers met de hand erin geschreven worden omdat anders de nummers van alle (141) interpolatiepunten er ook in terecht zouden zijn gekomen, wat een rommelig geheel opgeleverd zou hebben. (Dit voorbeeld laat zien dat een driedimensionaal programma voor curve-fitting ook een tweedimensionale kromme kan opleveren: omdat de tien gegeven punten in het yz-vlak liggen, ligt ook de gegenereerde kromme daarin.)

Wat de bovengenoemde wiskundige functies betreft, dit zijn derdegraads-polynomen die de kromme beschrijven in parametrische vorm:

$$\begin{aligned} x &= a_3t^3 + a_2t^2 + a_1t + a_0 \\ y &= b_3t^3 + b_2t^2 + b_1t + b_0 \\ z &= c_3t^3 + c_2t^2 + c_1t + c_0 \end{aligned} \qquad (3.2)$$

Omdat de kromme, zoals vermeld, uit m - 3 segmenten bestaat, zijn er ook m - 3 sets van derdegraadsfuncties (waarbij elke 'set' bestaat uit een functie voor x, één voor y en één voor z).

Voor elk segment van de kromme, dat ligt tussen de punten i en $i + 1$, neemt de variabele t waarden tussen 0 en 1 aan, dat wil zeggen voor $t = 0$ hebben we het ene eindpunt, dat in de buurt van punt i ligt, en voor $t = 1$ hebben we het andere, in de buurt van punt $i + 1$. Ook kunnen we voor zo'n segment, liggende tussen de punten i en $i + 1$, de coëfficiënten a_j, b_j, c_j ($j = 0, 1, 2, 3$) berekenen uit de coördinaten van vier naburige punten, namelijk de punten $i - 1$, i, $i + 1$ en $i + 2$; omdat we niet al te theoretisch op dit onderwerp in willen gaan, volgen hier uitdrukkingen voor de genoemde coëfficiënten zonder hun afleiding:

$$\begin{aligned} a_3 &= (-x_{i-1} + 3x_i - 3x_{i+1} + x_{i+2})/6 \\ a_2 &= (x_{i-1} - 2x_i + x_{i+1})/2 \\ a_1 &= (-x_{i-1} + x_{i+1})/2 \\ a_0 &= (x_{i-1} + 4x_i + x_{i+1})/6 \\[1em] b_3 &= (-y_{i-1} + 3y_i - 3y_{i+1} + y_{i+2})/6 \\ b_2 &= (y_{i-1} - 2y_i + y_{i+1})/2 \\ b_1 &= (-y_{i-1} + y_{i+1})/2 \\ b_0 &= (y_{i-1} + 4y_i + y_{i+1})/6 \\[1em] c_3 &= (-z_{i-1} + 3z_i - 3z_{i+1} + z_{i+2})/6 \\ c_2 &= (z_{i-1} - 2z_i + z_{i+1})/2 \\ c_1 &= (-z_{i-1} + z_{i+1})/2 \\ c_0 &= (z_{i-1} + 4z_i + z_{i+1})/6 \end{aligned} \qquad (3.3)$$

Let erop dat deze drie groepen van vier vergelijkingen dezelfde structuur hebben.

In (3.2) beschouwen we x, y en z als polynomen in t en in (3.3) vinden we de coëfficiënten van die polynomen. Terwille van de volledigheid en met het oog op wat we in paragraaf 3.9 zullen bespreken, behoort vermeld te worden dat als we in (3.2) de symbolen a_3 enzovoort vervangen door de corresponderende uitdrukkingen die in (3.3) zijn gegeven, we de resulterende uitdrukkingen opnieuw kunnen rangschikken, waardoor het volgende krijgen:

$$\begin{aligned} x &= F_{-1}(t)\, x_{i-1} + F_0(t)\, x_i + F_1(t)\, x_{i+1} + F_2(t)\, x_{i+2} \\ y &= F_{-1}(t)\, y_{i-1} + F_0(t)\, y_i + F_1(t)\, y_{i+1} + F_2(t)\, y_{i+2} \\ z &= F_{-1}(t)\, z_{i-1} + F_0(t)\, z_i + F_1(t)\, z_{i+1} + F_2(t)\, z_{i+2} \end{aligned} \qquad (3.4)$$

De volgende hierin voorkomende functies worden *mengfuncties* (in het Engels *blending functions*) genoemd:

$$\begin{aligned} F_{-1}(t) &= (-t^3 + 3t^2 - 3t + 1)/6 \\ F_0(t) &= (3t^3 - 6t^2 + 4)/6 \\ F_1(t) &= (-3t^3 + 3t^2 + 3t + 1)/6 \\ F_2(t) &= t^3/6 \end{aligned} \qquad (3.5)$$

In programma CURVE3 worden x, y en z berekend volgens (3.2), wat efficiënter is dan de ermee equivalente vormen (3.4). Laatstgenoemde laten evenwel duidelijker zien dat elk nieuw punt (x, y, z) berekend wordt als een lineaire combinatie van de gegeven vier punten die in de buurt liggen en dat de coëfficiënten (dat wil zeggen de mengfuncties) afhangen van t.

```
/* CURVE3: B-spline curve fitting in drie dimensies.
   Het programma leest een invoerfile en schrijft een
   uitvoerfile, die beide compatibel zijn met D3D, zodat
   we onze werkwijze als volgt schematisch kunnen weergeven:

D3D  --> input file  --> CURVE3  --> output file  --> D3D

   De invoerfile bevat regels met puntnummers i en driedimen-
   sionale coordinaten van de m punten P(i) (i = 1, ..., m).
   Een eventueel gedeelte in de file dat begint met het woord
   'Faces' wordt genegeerd. Het programma schrijft de driedimen-
   sionale coordinaten van de k punten Q(j) (j = 1, ..., k) in
   de uitvoerfile (met elk drietal coordinaten voorafgegaan
   door een puntnummer j), waarbij k = (m-3)N + 1.
```

```
   De waarde van N wordt ingelezen van het toetsenbord. (N is
   het aantal intervallen tussen twee opeenvolgende punten
   P(i) en P(i+1).

   Punt Q(1) is een benadering van P(2);
   punt Q(N+1) is een benadering van P(3);
   punt Q(2N+1) is een benadering van P(4);
     ...
   punt Q(k) is een benadering van P(m-1).
*/
#include <stdio.h>
#include <alloc.h>
#include <process.h>

main()
{ char infil[30], outfil[30];
  int m=0, k, N, i, j, first=0;
  float *x, *y, *z, X, Y, Z, t, xdum, ydum, zdum,
    xA, xB, xC, xD, yA, yB, yC, yD, zA, zB, zC, zD,
    a0, a1, a2, a3, b0, b1, b2, b3, c0, c1, c2, c3;
  FILE *fpin, *fpout;
  printf("Invoerfile:  "); scanf("%s", infil);
  printf("Uitvoerfile: "); scanf("%s", outfil);
  printf("Geef N, het aantal intervallen tussen\n");
  printf("twee opeenvolgende gegeven punten: ");
  scanf("%d", &N);
  fpin = fopen(infil, "r");
  if (fpin == NULL)
  { printf("File niet beschikbaar"); exit(1);
  }
  while (
  fscanf(fpin, "%*d %f %f %f", &xdum, &ydum, &zdum) >0) m++;
  fclose(fpin); fpin = fopen(infil, "r");
  x = (float *)malloc((m+1) * sizeof(float));
  y = (float *)malloc((m+1) * sizeof(float));
  z = (float *)malloc((m+1) * sizeof(float));
  if (z == NULL) {printf("Niet genoeg geheugen"); exit(1);}
  for (i=1; i<=m; i++)
    fscanf(fpin, "%*d %f %f %f", x+i, y+i, z+i);
  fclose(fpin);
  fpout = fopen(outfil, "w");
  for (i=2; i<m-1; i++)
  { xA=x[i-1]; xB=x[i]; xC=x[i+1]; xD=x[i+2];
    yA=y[i-1]; yB=y[i]; yC=y[i+1]; yD=y[i+2];
    zA=z[i-1]; zB=z[i]; zC=z[i+1]; zD=z[i+2];
```

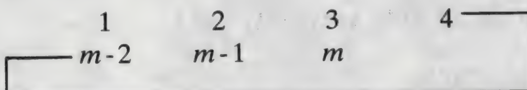
```
    a3=(-xA+3*(xB-xC)+xD)/6.0;
    a2=(xA-2*xB+xC)/2.0;
    a1=(xC-xA)/2.0;
    a0=(xA+4*xB+xC)/6.0;

    b3=(-yA+3*(yB-yC)+yD)/6.0;
    b2=(yA-2*yB+yC)/2.0;
    b1=(yC-yA)/2.0;
    b0=(yA+4*yB+yC)/6.0;

    c3=(-zA+3*(zB-zC)+zD)/6.0;
    c2=(zA-2*zB+zC)/2.0;
    c1=(zC-zA)/2.0;
    c0=(zA+4*zB+zC)/6.0;

    for (j=first; j<=N; j++)
    { t = (float)j/(float)N;
      X = ((a3*t+a2)*t+a1)*t+a0;
      Y = ((b3*t+b2)*t+b1)*t+b0;
      Z = ((c3*t+c2)*t+c1)*t+c0;
      fprintf(fpout, "%d %f %f %f\n", (i-2)*N+j+1, X, Y, Z);
    }
    first = 1;
  }
  fprintf(fpout, "Faces:\n"); k = (m-3)*N+1;
  for (j=1; j<k; j++) fprintf(fpout, "%d %d.\n", j, j+1);
  fclose(fpout);
}
```

Voordat we deze paragraaf beëindigen is het goed erop te wijzen dat we ook heel gemakkelijk een *gesloten* kromme kunnen verkrijgen. Daar een B-spline kromme gewoonlijk begint in de buurt van het tweede punt (punt 2) en eindigt bij het op een na laatste punt (punt m - 1), moeten we, om een gesloten kromme te verkrijgen, deze twee punten laten samenvallen. Verder moet dan punt 1 samenvallen met punt m - 2 en punt 3 met punt m, zoals schematisch weergegeven wordt door:



Dit betekent dus dat er twee samenvallende lijnstukken zijn.

Het is misschien ook de moeite waard op te merken dat twee of meer opeenvolgende punten mogen samenvallen. Het is hierdoor mogelijk een kromme

naar een gegeven punt 'toe te trekken'. Dit is vooral nuttig als we, in het geval van een open kromme, de eindpunten (de punten 1 en *m*) willen benaderen. We kunnen bijvoorbeeld het allereerste punt de nummers 1, 2 en 3 geven, waarmee we de kromme dwingen precies in dat punt te beginnen. Op het eerste gezicht mag dit vreemd lijken, omdat volgens de vergelijkingen (3.4) een berekend punt (x, y, z) wordt afgeleid van vier gegeven punten, dus we zijn geneigd te denken dat we alleen maar kunnen garanderen dat de kromme in punt 1 begint als het samenvalt met de punten 2, 3 *en* 4. Volgens de vergelijkingen (3.4) zou evenwel de coëfficiënt van punt 4 gelijk zijn aan $F_2(t)$, en (3.5) laat zien dat deze coëfficiënt gelijk aan nul is voor $t = 0$, wat voor het allereerste punt van de kromme inderdaad het geval is. Dus alleen de punten 1, 2 en 3 dragen effectief bij aan de positie van beginpunt van de kromme en als die drie punten in één punt samenvallen, dan zal de kromme in dat punt beginnen.

3.9 KABELS

Na onze voorbereidingen in de paragrafen 3.7 en 3.8 zijn we nu in staat een programma te ontwikkelen voor het genereren van 'kabels' zoals we die besproken hebben in paragraaf 2.5. Onze functies *initrotate* en *rotate*, besproken in paragraaf 3.7, zullen voor dit doel bijzonder nuttig blijken te zijn.

Bij een ruimtelijke kromme, gegeven als een rij punten die bij voorkeur gegenereerd zijn door programma CURVE3 (zie paragraaf 3.8), zullen we deze punten gebruiken als middelpunten van cirkels. De straal R van de cirkels wordt ingetypt, evenals n, het aantal punten op elke cirkel. (Zoals gebruikelijk, zal iedere cirkel worden benaderd door een regelmatige veelhoek met n hoekpunten.) De n hoekpunten van de eerste veelhoek zullen berekend worden op een manier die sterk afwijkt van de berekening van de hoekpunten van de andere veelhoeken. Laten we de eerste drie gegeven punten op de kromme aanduiden met C_0, C_1 en C_2. Punt C_1 zal dienen als middelpunt van de eerste cirkel, C_2 als dat van de tweede, enzovoort, dus er komt geen cirkel met het allereerste punt, C_0, als middelpunt.

We gaan eerst de punten C_0 en C_2 gebruiken om de as van de eerste cirkel (met middelpunt C_1) te bepalen. De vergelijking van het vlak α waarin deze cirkel ligt kan worden uitgedrukt in de coördinaten x_i, y_i, z_i van de drie punten C_i $(i = 0, 1, 2)$:

$$ax + by + cz = d$$

waarin

$$a = x_2 - x_0$$
$$b = y_2 - y_0$$
$$c = z_2 - z_0$$
$$d = ax_1 + by_1 + cz_1$$

Aangezien $\mathbf{n} = [a, b, c]$ de vector is die begint in C_0 en eindigt in C_2, is de lijn door C_1 met dezelfde richting als **n** de as van de cirkel waarin we geïnteresseerd zijn. We kunnen nu gemakkelijk een vector $\mathbf{r} = [r_x, r_y, r_z]$ vinden die loodrecht op deze as staat:

$$r_x = b, r_y = -a, r_z = 0 \text{ als } |a| \geq \delta \text{ of } |b| \geq \delta,$$

$$r_x = 0, r_y = c, r_z = -b \text{ als } |a| < \delta \text{ en } |b| < \delta,$$

waarin δ een klein positief getal, zoals bijvoorbeeld 10^{-5} is.

De vector **r** is zo gekozen dat zijn inwendig produkt met **n** gelijk is aan nul; bovenstaande logische beslissing voorkomt dat we een vector **r** krijgen waarvan de lengte (vrijwel) 0 is.

Na de berekening van

$$L = \sqrt{(r_x^2 + r_y^2 + r_z^2)},$$

kunnen we het punt met de volgende coördinaten gebruiken als het eerste punt van de cirkel in kwestie:

$$x[0] = x_1 + Rr_x/L$$
$$y[0] = y_1 + Rr_y/L$$
$$z[0] = z_1 + Rr_z/L$$

Daar we op iedere cirkel n punten willen hebben, gebruiken we

$$\theta = 2\pi/n$$

en vinden de overige n - 1 punten op de cirkel door iedere keer het voorgaande punt over een hoek θ te roteren om de as van de cirkel. Hoe ingewikkeld dit misschien ook lijkt, alles wat we te doen hebben is het aanroepen van de functies *initrotate* en *rotate*. Eerst roepen we *initrotate* aan, met, in deze volgorde, x_1, y_1, z_1, a, b, c, θ als argumenten en daarna voeren we de volgende for-statement uit:

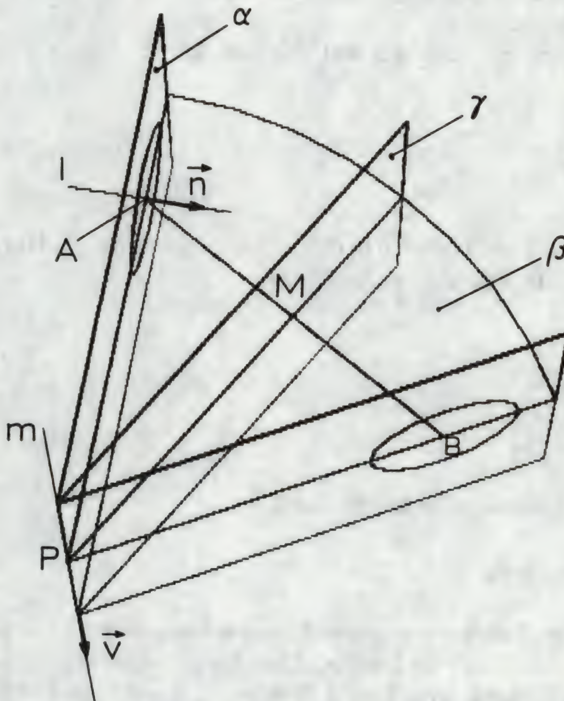
```
for (i=1; i<n; i++)
   rotate(x[i-1], y[i-1], z[i-1], x+i, y+i, z+i);
```

(Bedenk dat in de taal C de uitdrukking $x + i$ het adres van het array-element $x[i]$ oplevert, zodat we ook de uitdrukking &$x[i]$ ervoor in de plaats hadden mogen gebruiken.)

We moeten nu de overige cirkels berekenen en we doen dit door de n punten op

iedere nieuwe cirkel af te leiden uit overeenkomstige punten op de onmiddellijk eraan voorafgaande cirkel. Laten we de letter A gebruiken voor het middelpunt van deze voorafgaande cirkel (kortweg *cirkel A* genoemd) en B voor dat van de nieuwe cirkel (*cirkel B*). (Dus onmiddellijk nadat de allereerste cirkel geconstrueerd is, hebben we $A = C_1$ en $B = C_2$.) Laten we de as van cirkel A aanduiden door de letter l. Als B op l ligt, dan kunnen we de n punten van cirkel B verkrijgen door een eenvoudige translatie van de corresponderende punten op cirkel A; we hoeven dan alleen maar $x_B - x_A$, $y_B - y_A$, en $z_B - z_A$ op te tellen bij de coördinaten van de n punten op cirkel A.

De zaken liggen niet zo eenvoudig als B buiten l ligt. In dat geval kunnen we cirkel B verkrijgen door de n punten van cirkel A te roteren, waarbij de hierbij ontstane beeldpunten de gewenste punten op cirkel B zijn. We kunnen nu nogmaals onze functies *initrotate* en *rotate* gebruiken, mits we een as en een hoek van rotatie kunnen leveren. Let erop dat deze rotatie essentieel verschilt van de rotatie die we hebben gebruikt om $n - 1$ punten op de eerste cirkel te vinden. We beschouwen drie vlakken, aangegeven in figuur 3.17, namelijk:



Figuur 3.17 Het roteren van cirkel B te verkrijgen

vlak α, waarin cirkel A ligt;
vlak β, dat gaat door lijn l en punt B;
vlak γ, dat gaat door het midden M van lijnstuk AB en dat loodrecht op dat lijnstuk staat.

Voor vlak α hebben we de vergelijking

$$ax + by + cz = d,$$

waarbij we de eerste keer gebruik maken van de waarden a, b, c en d, die we hebben berekend voor de eerste driehoek (met middelpunt C_1). Het zal tegen het eind van deze paragraaf ook duidelijk worden hoe we de volgende keren aan deze vier coëfficiënten komen.

Voor vlak β gebruiken we een parametervoorstelling in plaats van een vergelijking. We weten dat dit vlak gaat door punt A en dat het evenwijdig is aan de twee vectoren $[a, b, c]$ en **AB**. Dit leidt tot de volgende representatie van vlak β, waarin λ en μ parameters zijn:

$$[x\ y\ z] = [x_A\ y_A\ z_A] + \lambda[a\ b\ c] + \mu[x_B\text{-}x_A\ y_B\text{-}y_A\ z_B\text{-}z_A]$$

Vlak γ staat loodrecht op vector **AB**, wat betekent dat we

$$d_x = x_B - x_A$$
$$d_y = y_B - y_A$$
$$d_z = z_B - z_A$$

kunnen gebruiken als coëfficiënten in de vergelijking van vlak α. Het gaat door het midden M van AB. Dus gebruik makend van

$$x_M = (x_A + x_B)/2$$
$$y_M = (y_A + y_B)/2$$
$$z_M = (z_A + z_B)/2$$
$$d_0 = d_x x_M + d_y y_M + d_z z_M,$$

vinden we de volgende vergelijking voor vlak γ:

$$d_x x + d_y y + d_z z = d_0$$

De vlakken α, β en γ hebben één snijpunt, dat we P noemen; we kunnen dit punt nu berekenen. De lijn m door P en loodrecht op vlak β is dan de as van rotatie die we nodig hebben en de hoek APB (dat is de hoek tussen de lijnen AP en BP) is de benodigde hoek van rotatie. We zullen voor deze hoek de letter φ gebruiken. Eigenlijk moeten we een *gerichte* as hebben, met andere woorden, behalve punt P

hebben we een vector v nodig, zodanig dat als we ons voorstellen dat v op lijn *m* ligt, we cirkel C_1 over hoek φ om v in positieve zin moeten roteren (waarbij de draaiing met de richting van v correspondeert op de manier van een schroefbeweging). We vinden deze vector v als het vectorprodukt (ofwel 'uitwendig' produkt) van de vectoren **PA** en **PB**:

$$\mathbf{v} = \mathbf{PA} \times \mathbf{PB}$$

We kunnen de vectoren **PA** en **PB** ook gebruiken om de hoek φ af te leiden uit hun inwendig produkt

$$\mathbf{PA} \,.\, \mathbf{PB} = PA \,.\, PB \,.\, \cos\varphi,$$

waarin PA en PB de lengten van de vectoren **PA** en **PB** voorstellen.

Nadat we cirkel B door rotatie uit cirkel A hebben verkregen, moeten we niet vergeten ook een nieuwe vector [*a b c*], loodrecht op cirkel B te bepalen, aangezien in de volgende stap cirkel B zal optreden als de oude cirkel A en we nieuwe waarden *a*, *b* en *c* nodig zullen hebben. Hoewel het vaak prettig is het beginpunt van een vector elders te kiezen, plaatsen we het nu in de oorsprong O van het coördinatenstelsel; we zijn dan geïnteresseerd in het nieuwe punt (*a*', *b*', *c*'), verkregen door punt (*a*, *b*, *c*) te roteren om vector v, die deze keer eveneens O als beginpunt heeft. De nieuwe waarden van de programmavariabelen *a*, *b* en *c* worden zodoende als volgt verkregen:

```
initrotate(0.0, 0.0, 0.0, v1, v2, v3, phi);
rotate(a, b, c, &a, &b, &c);
```

Verdere bijzonderheden omtrent de berekening van P, v en φ zijn te vinden in de programmatekst zelf:

```
/* CABLE: Dit programma leest een file met gegevens over
          een ruimtelijke kromme en genereert een file met
          gegevens omtrent een hieruit ontstane kabel.
          Als dwarsdoorsnede van de kabel gebruiken
          we een regelmatige veelhoek met n hoekpunten,
          die een cirkel met straal R benadert. Zowel
          n als R worden van het toetsenbord ingelezen.
          Het programma moet door de 'linker' worden samen-
          gevoegd met de module TRAFO, waarin de functies
          'initrotate' en 'rotate' zijn gedefinieerd.
*/
#include <stdio.h>
```

```
#include <math.h>
#include <alloc.h>
#include <process.h>

void initrotate(double a1, double a2, double a3,
          double v1, double v2, double v3, double alpha);
void rotate(double x, double y, double z,
       double *px1, double *py1, double *pz1);

int zero(double x);

double *getdouble(int n);

double *enlarge(double *px, int N);

void ermes(char *s);

main()
{ char infil[30], outfil[30];
  int i, n, j, m, jn, jn0, k, tablesize;
  double R, *x, *y, *z, xC0, yC0, zC0, xC1, yC1, zC1,
    xC2, yC2, zC2, *xC, *yC, *zC,
    a, b, c, d, rx, ry, rz, pi, theta, Len,
    *getdouble(), *enlarge(),
    xA, yA, zA, xB, yB, zB, dx, dy, dz, d0, c1, c2, c0,
    xM, yM, zM, e1, e2, e0, denom, lambda, mu, xP, yP, zP,
    xAP, yAP, zAP, xBP, yBP, zBP, v1, v2, v3,
    cosphi, phi;
  FILE *fpin, *fpout;
  printf("Invoerfile:  "); scanf("%s", infil);
  fpin = fopen(infil, "r");
  if (fpin == NULL)
     ermes("File niet beschikbaar");
  if
  (fscanf(fpin, "%*d %lf %lf %lf", &xC0, &yC0, &zC0) != 3 ||
   fscanf(fpin, "%*d %lf %lf %lf", &xC1, &yC1, &zC1) != 3 ||
   fscanf(fpin, "%*d %lf %lf %lf", &xC2, &yC2, &zC2) != 3)
       ermes("Invoerfile incorrect");
  printf("Uitvoerfile: "); scanf("%s", outfil);
  fpout = fopen(outfil, "w");
  printf("Hoeveel punten op iedere cirkel? ");
  scanf("%d", &n);
  printf("Straal: "); scanf("%lf", &R);
  a=xC2-xC0; b=yC2-yC0; c=zC2-zC0; d=a*xC1+b*yC1+c*zC1;
```

```
/* De eerste cirkel heeft middelpunt (xC1, yC1, zC1),
   straal R en hij ligt in vlak ax + by + cz = d
*/
x = getdouble(n); y = getdouble(n); z = getdouble(n);
if (zero(a) && zero(b)) {rx=0; ry=c; rz=-b;}
                   else {rx=b; ry=-a; rz=0;}
Len=sqrt(rx*rx+ry*ry+rz*rz);
rx/=Len; ry/=Len; rz/=Len;
/* (rx, ry, rz) is een eenheidsvector loodrecht op
   (a, b, c)
*/
x[0]=xC1+rx*R; y[0]=yC1+ry*R; z[0]=zC1+rz*R;
pi=4.0*atan(1.0);
theta=2*pi/n;
/* Berekening van n punten op de eerste cirkel: */
initrotate(xC1, yC1, zC1, a, b, c, theta);
for (i=1; i<n; i++)
  rotate(x[i-1], y[i-1], z[i-1], x+i, y+i, z+i);
/* Tel het aantal cirkels
   (1 minder dan het aantal gelezen punten):
*/
m = 2;
while (
fscanf(fpin, "%*d %lf %lf %lf", &xC0, &yC0, &zC0) == 3)
                                                   m++;
tablesize = m*n;
x = enlarge(x, tablesize);
y = enlarge(y, tablesize);
z = enlarge(z, tablesize);
fclose(fpin); fpin = fopen(infil, "r");       /* Rewind */
fscanf(fpin, "%*d %lf %lf %lf", &xC0, &yC0, &zC0);
                                                /* Skip */
xC = getdouble(m);
yC = getdouble(m);
zC = getdouble(m);
for (j=0; j<m; j++)
  fscanf(fpin, "%*d %lf %lf %lf", xC+j, yC+j, zC+j);
/* (xC[0], yC[0], zC[0]) is nu het middelpunt van de
   gegeven cirkel, die ligt in vlak ax + by + cz = d
   en de straal R heeft. De n relevante punten op
   deze cirkel zijn al berekend; hun coordinaten zijn
   x[0], y[0], z[0], ..., x[n-1], y[n-1], z[n-1].
   De andere m-1 cirkels zullen door middel van rotaties
   van deze eerste worden afgeleid.
```

```
*/
fclose(fpin);
for (j=1; j<m; j++)
{ jn=j*n; jn0=jn-n;
  xA=xC[j-1]; yA=yC[j-1]; zA=zC[j-1];
  xB=xC[j];  yB=yC[j];  zB=zC[j];
  dx=xB-xA; dy=yB-yA; dz=zB-zA;
  c1=a*a+b*b+c*c;
  c2=a*dx+b*dy+c*dz;
  c0=d-a*xA-b*yA-c*zA;
  xM=0.5*(xA+xB); yM=0.5*(yA+yB); zM=0.5*(zA+zB);
  d0=dx*xM+dy*yM+dz*zM;
  e1=dx*a+dy*b+dz*c;
  e2=dx*dx+dy*dy+dz*dz;
  e0=d0-dx*xA-dy*yA-dz*zA;
  denom=c1*e2-c2*e1;
  if (fabs(denom) < 1e-12)
  { /* Richting verandert niet.
       In plaats van een oneindig ver weg gelegen
       punt P te gebruiken, voeren we een eenvoudige
       translatie uit:
    */
    for (i=0; i<n; i++)
    { x[jn+i] = x[jn0+i] + dx;
      y[jn+i] = y[jn0+i] + dy;
      z[jn+i] = z[jn0+i] + dz;
    }
  } else
    /* Richting verandert.
       De veelhoek wordt geroteerd over de hoek phi
       om vector v die door punt P gaat:
    */
  { lambda=(c0*e2-c2*e0)/denom;
    mu=(c1*e0-c0*e1)/denom;
    xP=xA+lambda*a+mu*dx;
    yP=yA+lambda*b+mu*dy;
    zP=zA+lambda*c+mu*dz;
    /* Het snijpunt P van drie vlakken ligt op de
       as van rotatie.
    */
    xAP=xA-xP; yAP=yA-yP; zAP=zA-zP;
    xBP=xB-xP; yBP=yB-yP; zBP=zB-zP;
    v1=yAP*zBP-yBP*zAP;
    v2=xBP*zAP-xAP*zBP;
```

```
      v3=xAP*yBP-xBP*yAP;
      /* (v1, v2, v3) is richting van rotatie-as */
      cosphi=(xAP*xBP+yAP*yBP+zAP*zBP)/
             sqrt((xAP*xAP+yAP*yAP+zAP*zAP)*
                  (xBP*xBP+yBP*yBP+zBP*zBP));
      phi = (cosphi == 0 ? 0.5*pi :
           atan(sqrt(1.0-cosphi*cosphi)/cosphi));
        /* phi is de rotatiehoek */
        initrotate(xP, yP, zP, v1, v2, v3, phi);
        for (i=0; i<n; i++)
          rotate(x[jn0+i], y[jn0+i], z[jn0+i],
               x+jn+i, y+jn+i, z+jn+i);
        initrotate(0.0, 0.0, 0.0, v1, v2, v3, phi);
        rotate(a, b, c, &a, &b, &c);
      }
      d=a*xC[j]+b*yC[j]+c*zC[j];
    }
    if (fpout == NULL) ermes("Probleem met uitvoerfile");
    for (k=0; k<m*n; k++)
      fprintf(fpout, "%d %f %f %f\n", k+1, x[k], y[k], z[k]);
    fprintf(fpout, "Faces:\n");

    for (k=n-1; k>=0; k--) fprintf(fpout, " %d", k+1);
    fprintf(fpout, ".\n");

    for (k=(m-1)*n; k<m*n; k++) fprintf(fpout, " %d", k+1);
    fprintf(fpout, ".\n\n");

    for (j=1; j<m; j++)
    { jn=j*n+1; jn0=jn-n;
      for (i=0; i<n-1; i++)
      { fprintf(fpout, " %d %d %d.\n", jn+i+1, -(jn0+i),
          jn0+i+1);
        fprintf(fpout, " %d %d %d.\n", jn0+i, -(jn+i+1),
          jn+i);
      }
      fprintf(fpout, " %d %d %d.\n\n", jn, -(jn0+n-1), jn0);
      fprintf(fpout, " %d %d %d.\n\n", jn0+n-1, -jn, jn+n-1);
    }
    fclose(fpout);
}

int zero(double x)
{ return fabs(x) < 1e-5;
}
```

```
double *getdouble(int n)
{ double *p;
  p = (double *)malloc(n * sizeof(double));
  if (p == NULL) ermes("Niet genoeg geheugen");
  return p;
}

double *enlarge(double *px, int N)
{ px = (double *)realloc(px, N*sizeof(double));
  if (px == NULL) ermes("Niet genoeg geheugen");
  return px;
}

void ermes(char *s)
{ printf(s); exit(1);
}
```

Voor demonstraties van programma CABLE kan volstaan worden met te verwijzen naar paragraaf 2.5, zie de figuren 2.16, 2.19 en 2.20.

3.10 B-SPLINE OPPERVLAKKEN

In paragraaf 3.8 hebben we ons beziggehouden met ruimtelijke krommen, waarvan de punten de coördinaten $x(t)$, $y(t)$, $z(t)$ hebben; hierin werd gebruik gemaakt van een parameter t. We zullen ons nu gaan bezighouden met oppervlakken. De punten hiervan hebben de coördinaten $x(u, v)$, $y(u, v)$, $z(u, v)$, waarin u en v twee onafhankelijke parameters zijn. We zullen ook nu gebruik maken van de B-spline approximatiemethode. Afgezien van het feit dat er nu twee parameters zijn, is de situatie analoog aan die bij B-spline krommen. In plaats van een rij van m gegeven punten, zijn nu nm punten gegeven, waarvan we de nummers als volgt in een tabel kunnen plaatsen:

$$
\begin{array}{cccc}
P_{11} & P_{12} & \dots & P_{1m} \\
P_{21} & P_{22} & \dots & P_{2m} \\
. & . & \dots & . \\
. & . & \dots & . \\
. & . & \dots & . \\
P_{n1} & P_{n2} & \dots & P_{nm}
\end{array}
$$

We zullen de begrippen *horizontaal* en *verticaal* gebruiken met betrekking tot de rijen en kolommen van deze tabel; we kunnen bijvoorbeeld zeggen dat P_{1m} het meest rechtse punt op de eerste horizontale lijn is.

Het programma dat we gaan bespreken leest een gewone objectfile in D3D-formaat, waarin de punten gegeven zijn in bovenstaande volgorde P_{11}, P_{12}, ...P_{nm}. Het programma leest m en n van het toetsenbord en verwacht dan een invoerfile waarin de punten doorlopend genummerd zijn van 1 tot mn. Dus punt P_{ij} in bovenstaande tabel heeft in werkelijkheid nummer

$$k = (i - 1)m + j$$

in de invoerfile.

We kunnen bovenstaande tabel van punten P_{ij} ook opvatten als twee stelsels krommen: voor iedere 'horizontale' kromme is i constant en loopt j van 1 tot m en voor iedere 'verticale' kromme is j constant en loopt i van 1 tot n. Zoals we weten worden bij B-spline approximatie het allereerste en het allerlaatste punt van een kromme weliswaar gebruikt in de berekening maar niet benaderd. Hetzelfde is van toepassing op de punten P_{ij} in bovenstaande tabel waarvoor geldt $i = 1$ of $i = n$ en op die waarvoor geldt $j = 1$ of $j = m$. Voor de berekening van de roosterpunten van een 'rechthoek' (vergelijkbaar met een 'segment' van een kromme) hebben we nu vier punten in beide richtingen nodig, wat tezamen neerkomt op de volgende 16 punten:

$P_{i-1,j-1}$	$P_{i-1,j}$	$P_{i-1,j+1}$	$P_{i-1,j+2}$
$P_{i,j-1}$	$P_{i,j}$	$P_{i,j+1}$	$P_{i,j+2}$
$P_{i+1,j-1}$	$P_{i+1,j}$	$P_{i+1,j+1}$	$P_{i+1,j+2}$
$P_{i+2,j-1}$	$P_{i+2,j}$	$P_{i+2,j+1}$	$P_{i+2,j+2}$

Al deze punten zullen worden gebruikt om de 'rechthoek' in het midden (met $P_{i,j}$ in zijn hoekpunt linksboven en $P_{i+1,j+1}$ in zijn hoekpunt rechtsonder) te benaderen. Deze rechthoek in het midden zal worden verdeeld in $N \times M$ kleine oppervlak-elementen (zodanig dat we er N in verticale en M in horizontale richting tellen). Het programma zal M en N inlezen van het toetsenbord.

Gebruik makend van de formules voor de vier functies F_{-1}, F_0, F_1 en F_2, gegeven in (3.5) (zie paragraaf 3.8), vervangen we de formules (3.4) door overeenkomstige formules voor de tweedimensionale situatie; omdat deze voor de drie variabelen x, y en z dezelfde structuur hebben is alleen die voor x op de volgende bladzijde weergegeven:

$$
\begin{aligned}
x = {} & F_{-1}(u)F_{-1}(v)x_{i-1,i-1} + F_{-1}(u)F_0(v)x_{i-1,i} + \\
& F_{-1}(u)F_1(v)x_{i-1,i+1} + F_{-1}(u)F_2(v)x_{i-1,i+1} + \\
\\
& F_0(u)F_{-1}(v)x_{i,i-1} + F_0(u)F_0(v)x_{i,i} + \\
& F_0(u)F_1(v)x_{i,i+1} + F_0(u)\,F_2(v)x_{i,i+1} + \qquad (3.6) \\
\\
& F_1(u)F_{-1}(v)x_{i+1,i-1} + F_1(u)F_0 \\
& F_1(u)F_1(v)x_{i+1,i+1} + F_1(u)F_2(v)x_{i+1,i+1} + \\
\\
& F_2(u)F_{-1}(v)x_{i+2,i-1} + F_2(u)F_0(v)x_{i+2,i} + \\
& F_2(u)F_1(v)x_{i+2,i+1} + F_2(u)F_2(v)x_{i+2,i+1}
\end{aligned}
$$

Zoals besproken in paragraaf 3.8, hebben we daar (3.4) niet echt gebruikt, maar in plaats ervan de equivalente vorm (3.2), die polynomen in t bevat. We kunnen hier hetzelfde doen en (3.6) op een zodanige manier herschrijven dat we sommen van zestien termen krijgen, waarin de volgende produkten voorkomen:

$$
u^3v^3, u^3v^2, u^3v, u^3, u^2v^3, u^2v^2, u^2v, u^2, uv^3, uv^2, uv, u, \\
v^3, v^2, v, 1
$$

In deze termen zijn de coëfficiënten nogal ingewikkelde formules, die ik hier weglaat om duplicering van werk te voorkomen: zij zijn gebruikt in programma BSPLSURF en zijn daar gemakkelijk te vinden.

```
/* BSPLSURF: B-spline oppervlakken.
   Dit programma verwacht een D3D-objectfile waarin een rooster
   van punten is gedefinieerd. Het programma vraagt aan de ge-
   bruiker m en n in te typen: er moeten precies mn punten in
   de file aanwezig zijn, genummerd 1, 2, ..., mn. Aangenomen
   wordt dat deze punten als volgt in een rooster zijn gerang-
   schikt:

          1          2       ...      m
         m+1        m+2      ...     2m
        2m+1       2m+2      ...     3m
          .          .       ...      .
          .          .       ...      .
          .          .       ...      .
     (n-1)m+1    (n-1)+2     ...     mn
```

```
   Zowel m als n moeten ten minste gelijk aan 4 zijn.
   (De positie van de punten in de driedimensionale ruimte
   is willekeurig: voor twee punten die in dit rooster
   naast of onder elkaar staan hoeft niet x, y of z gelijk
   te zijn.)
   Als de objectfile een sectie bevat die begint met "Faces:",
   dan wordt die sectie genegeerd.
   Er moeten ook twee getallen M en N worden ingetypt. Zij
   geven het aantallen intervallen aan die in het benaderings-
   proces gebruikt worden. Er zullen M intervallen zijn tussen
   de punten 1 en 2; evenzo zijn er N intervallen tussen de
   punten 1 en m+1, enzovoort. Het B-spline gekromde oppervlak
   dat de gegeven punten benadert wordt in D3D-formaat ge-
   schreven naar een uitvoerfile.
*/
#include <stdio.h>
#include <process.h>
#include <alloc.h>
main()
{ FILE *fp1, *fp2;
  char str[30];
  float *xx, *yy, *zz, x, y, z, v, u, u2, u3,
    a00, a01, a02, a03,
    a10, a11, a12, a13,
    a20, a21, a22, a23,
    a30, a31, a32, a33,
    b00, b01, b02, b03,
    b10, b11, b12, b13,
    b20, b21, b22, b23,
    b30, b31, b32, b33,
    c00, c01, c02, c03,
    c10, c11, c12, c13,
    c20, c21, c22, c23,
    c30, c31, c32, c33,
    x00, x01, x02, x03,
    x10, x11, x12, x13,
    x20, x21, x22, x23,
    x30, x31, x32, x33,
    y00, y01, y02, y03,
    y10, y11, y12, y13,
    y20, y21, y22, y23,
    y30, y31, y32, y33,
    z00, z01, z02, z03,
    z10, z11, z12, z13,
```

```
  z20, z21, z22, z23,
  z30, z31, z32, z33;

int size, i, j, m, n, M, N, mn, l, k, I, J,
  A, B, C, ii, jj, nn, mm, P,
  h00, h01, h02, h03,
  h10, h11, h12, h13,
  h20, h21, h22, h23,
  h30, h31, h32, h33;

do
{ printf("Geef m en n (beide ten minste 4): ");
  scanf("%d %d", &m, &n);
} while (m < 4 || n < 4);
printf("Invoerfile:    "); scanf("%s", str);
fp1 = fopen(str, "r");
printf("Uitvoerfile:   "); scanf("%s", str);
fp2 = fopen(str, "w");
if (fp1 == NULL || fp2 == NULL)
{ printf("File-probleem"); exit(1);
}
mn = m * n; size = (mn+1) * sizeof(float);
xx = malloc(size); yy = malloc(size); zz = malloc(size);
for (l=0; l<mn; l++)
{ if (fscanf(fp1, "%d %f %f %f", &k, &x, &y, &z) != 4)
  { printf("Te weinig punten in file"); exit(1);
  }
  if (k < 1 || k > mn)
  { printf("Punt nummer %d incorrect", k); exit(1);
  }
  xx[k] = x; yy[k] = y; zz[k] = z;
}
fclose(fp1);
printf("Geef M en N: "); scanf("%d %d", &M, &N);
/* Laten we ons voorstellen dat we n rijen van m
   punten hebben, die (n-1)(m-1) rechthoeken op-
   leveren. Hiervan zullen maar (n-3)(m-3) recht-
   hoeken worden benaderd. We stellen ons voor dat
   elk van deze rechthoeken wordt verdeeld in M x N
   heel kleine rechthoeken, die we 'elementen' noe-
   men. Er komen voor elke horizontale roosterlijn
   A punten in de uitvoerfile:
*/
A = (m-3) * M + 1;
```

```
printf("  i   j\n");
for (i=2; i<=n-2; i++)
for (j=2; j<=m-2; j++)
{ h11 = (i-1) * m + j;
  h10 = h11-1; h12 = h10+2; h13 = h10+3;
  h00 = h10-m; h01 = h00+1; h02 = h00+2; h03 = h00+3;
  h20 = h10+m; h21 = h20+1; h22 = h20+2; h23 = h20+3;
  h30 = h20+m; h31 = h30+1; h32 = h30+2; h33 = h30+3;

  printf("%3d %3d\n", i,  j);
                      /* Laat zien dat we bezig zijn */

  /* De 16 punten worden hier als volgt genummerd:

  P00  P01  P02  P03
  P10  P11  P12  P13
  P20  P21  P22  P23
  P30  P31  P32  P33

  Het binnenste gebied, tussen P11, P12, P21, P22, zal
  in deze stap (met de huidige waarde van i en j) wor-
  den benaderd.
  */

  x00 = xx[h00]; y00 = yy[h00]; z00 = zz[h00];
  x01 = xx[h01]; y01 = yy[h01]; z01 = zz[h01];
  x02 = xx[h02]; y02 = yy[h02]; z02 = zz[h02];
  x03 = xx[h03]; y03 = yy[h03]; z03 = zz[h03];

  x10 = xx[h10]; y10 = yy[h10]; z10 = zz[h10];
  x11 = xx[h11]; y11 = yy[h11]; z11 = zz[h11];
  x12 = xx[h12]; y12 = yy[h12]; z12 = zz[h12];
  x13 = xx[h13]; y13 = yy[h13]; z13 = zz[h13];

  x20 = xx[h20]; y20 = yy[h20]; z20 = zz[h20];
  x21 = xx[h21]; y21 = yy[h21]; z21 = zz[h21];
  x22 = xx[h22]; y22 = yy[h22]; z22 = zz[h22];
  x23 = xx[h23]; y23 = yy[h23]; z23 = zz[h23];

  x30 = xx[h30]; y30 = yy[h30]; z30 = zz[h30];
  x31 = xx[h31]; y31 = yy[h31]; z31 = zz[h31];
  x32 = xx[h32]; y32 = yy[h32]; z32 = zz[h32];
  x33 = xx[h33]; y33 = yy[h33]; z33 = zz[h33];
```

```
a33 = (x00-x03-x30+x33
      +3*(-x01+x02-x10+x13+x20-x23+x31-x32)
      +9*(x11-x12-x21+x22))/36;


a32 = (-x00-x02+x30+x32 + 2*(x01-x31)
      + 3*(x10+x12-x20-x22) + 6*(-x11+x21))/12;

a31 = (x00-x02-x30+x32 + 3*(-x10+x12+x20-x22))/12;

a30 = (-x00-x02+x30+x32 + 3*(x10+x12-x20-x22)
      + 4*(-x01+x31) + 12*(x11-x21))/36;

a23 = (-x00+x03-x20+x23 + 2*(x10-x13)
      + 3*(x01-x02+x21-x22) + 6*(x12-x11))/12;

a22 = (x00+x02+x20+x22 + 2*(-x01-x10-x12-x21))/4 + x11;

a21 = (x02-x00-x20+x22 + 2*(x10-x12))/4;

a20 = (x00+x02+x20+x22 - 2*(x10+x12)
      + 4*(x01+x21) - 8*x11)/12;

a13 = (x00-x03-x20+x23 + 3*(x02-x01+x21-x22))/12;

a12 = (-x00-x02+x20+x22 + 2*(x01-x21))/4;

a11 = (x00-x02-x20+x22)/4;

a10 = (-x00-x02+x20+x22 + 4*(-x01+x21))/12;

a03 = (x03-x00-x20+x23 + 3*(x01-x02+x21-x22)
      + 4*(x13-x10) + 12*(x11-x12))/36;

a02 = (x00+x02+x20+x22 - 2*(x01+x21)
      + 4*(x10+x12) - 8*x11)/12;

a01 = (x02-x00-x20+x22 + 4*(x12-x10))/12;

a00 = (x00+x02+x20+x22 + 4*(x01+x10+x12+x21)
      + 16*x11)/36;


b33 = (y00-y03-y30+y33
      +3*(-y01+y02-y10+y13+y20-y23+y31-y32)
```

```
        +9*(y11-y12-y21+y22))/36;

b32 = (-y00-y02+y30+y32 + 2*(y01-y31)
      + 3*(y10+y12-y20-y22) + 6*(-y11+y21))/12;

b31 = (y00-y02-y30+y32 + 3*(-y10+y12+y20-y22))/12;

b30 = (-y00-y02+y30+y32 + 3*(y10+y12-y20-y22)
      + 4*(-y01+y31) + 12*(y11-y21))/36;

b23 = (-y00+y03-y20+y23 + 2*(y10-y13)
      + 3*(y01-y02+y21-y22) + 6*(y12-y11))/12;

b22 = (y00+y02+y20+y22 + 2*(-y01-y10-y12-y21))/4 + y11;

b21 = (y02-y00-y20+y22 + 2*(y10-y12))/4;

b20 = (y00+y02+y20+y22 - 2*(y10+y12)
      + 4*(y01+y21) - 8*y11)/12;

b13 = (y00-y03-y20+y23 + 3*(y02-y01+y21-y22))/12;

b12 = (-y00-y02+y20+y22 + 2*(y01-y21))/4;

b11 = (y00-y02-y20+y22)/4;

b10 = (-y00-y02+y20+y22 + 4*(-y01+y21))/12;

b03 = (y03-y00-y20+y23 + 3*(y01-y02+y21-y22)
      + 4*(y13-y10) + 12*(y11-y12))/36;

b02 = (y00+y02+y20+y22 - 2*(y01+y21)
      + 4*(y10+y12) - 8*y11)/12;

b01 = (y02-y00-y20+y22 + 4*(y12-y10))/12;

b00 = (y00+y02+y20+y22 + 4*(y01+y10+y12+y21)
      + 16*y11)/36;


c33 = (z00-z03-z30+z33
      +3*(-z01+z02-z10+z13+z20-z23+z31-z32)
      +9*(z11-z12-z21+z22))/36;
```

```
c32 = (-z00-z02+z30+z32 + 2*(z01-z31)
      + 3*(z10+z12-z20-z22) + 6*(-z11+z21))/12;

c31 = (z00-z02-z30+z32 + 3*(-z10+z12+z20-z22))/12;

c30 = (-z00-z02+z30+z32 + 3*(z10+z12-z20-z22)
      + 4*(-z01+z31) + 12*(z11-z21))/36;

c23 = (-z00+z03-z20+z23 + 2*(z10-z13)
      + 3*(z01-z02+z21-z22) + 6*(z12-z11))/12;

c22 = (z00+z02+z20+z22 + 2*(-z01-z10-z12-z21))/4 + z11;

c21 = (z02-z00-z20+z22 + 2*(z10-z12))/4;

c20 = (z00+z02+z20+z22 - 2*(z10+z12)
      + 4*(z01+z21) - 8*z11)/12;

c13 = (z00-z03-z20+z23 + 3*(z02-z01+z21-z22))/12;

c12 = (-z00-z02+z20+z22 + 2*(z01-z21))/4;

c11 = (z00-z02-z20+z22)/4;

c10 = (-z00-z02+z20+z22 + 4*(-z01+z21))/12;

c03 = (z03-z00-z20+z23 + 3*(z01-z02+z21-z22)
      + 4*(z13-z10) + 12*(z11-z12))/36;

c02 = (z00+z02+z20+z22 - 2*(z01+z21)
      + 4*(z10+z12) - 8*z11)/12;

c01 = (z02-z00-z20+z22 + 4*(z12-z10))/12;

c00 = (z00+z02+z20+z22 + 4*(z01+z10+z12+z21)
      + 16*z11)/36;

/* In de uitvoerfile zullen we puntnummers k gebruiken,
   die moeten worden berekend uit de lopende variabelen
   i, j, I, J.
   In iedere rechthoek, die bepaald wordt door het paar
   (i, j) en die in N x M elementen moet worden verdeeld,
   gebruiken we I, tellend tot N, en J, tellend tot M.
   Om duplicatie van punten te voorkomen beginnen I en J
```

```
      normaal bij 1, behalve voor de laagste waarde (2) van
      i en j: dan beginnen I, respectievelijk J, bij 0.
      (Van alle rechthoeken die in elementen moeten worden
      verdeeld is die met i=j=2 de rechthoek helemaal links
      bovenaan.) Als I=0 (wat i=2 impliceert), dan is
      k = (j-2)M + J + 1 en anders is
      k = A + (i-2)NA + (I-1)A + (j-2)M + J + 1.
      Dit betekent dat in beide gevallen geldt:

        k = {(i-2)N + I)A + {(j-2)M + 1} + J

      (A is het totale aantal gebruikte punten dat we op een
       horizontale rij.)
   */
   B = (j-2)*M+1;
   for (I = (i == 2 ? 0 : 1); I <= N; I++)
   { u = (float)I/N; u2 = u*u; u3 = u2*u;
     C = ((i-2)*N+I)*A + B;
     for (J = (j == 2 ? 0 : 1); J <= M; J++)
     { k = C + J;
       v = (float)J/M;
       x = ((a03*v+a02)*v+a01)*v+a00+
           u*(((a13*v+a12)*v+a11)*v+a10)+
           u2*(((a23*v+a22)*v+a21)*v+a20)+
           u3*(((a33*v+a32)*v+a31)*v+a30);

       y = ((b03*v+b02)*v+b01)*v+b00+
           u*(((b13*v+b12)*v+b11)*v+b10)+
           u2*(((b23*v+b22)*v+b21)*v+b20)+
           u3*(((b33*v+b32)*v+b31)*v+b30);

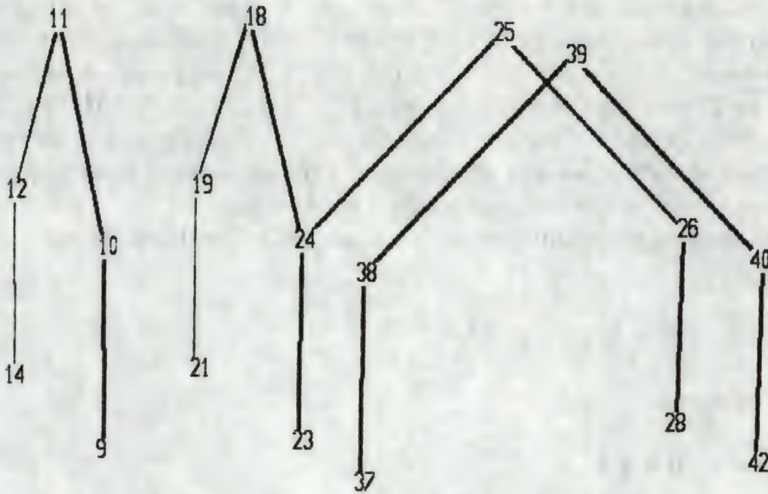
       z = ((c03*v+c02)*v+c01)*v+c00+
           u*(((c13*v+c12)*v+c11)*v+c10)+
           u2*(((c23*v+c22)*v+c21)*v+c20)+
           u3*(((c33*v+c32)*v+c31)*v+c30);
       fprintf(fp2, "%d %f %f %f\n", k, x, y, z);
     }
   }
}
fprintf(fp2, "Faces:\n");
nn = (n - 3) * N;
mm = A - 1;          /* mm = (m - 3) * M */
for (ii=0; ii<nn; ii++)
for (jj=0; jj<mm; jj++)
```

```
   { P = ii * A + jj + 1;
     /* We moeten ieder (bijna rechthoekig) element verdelen
        in twee driehoeken om er zeker van te zijn dat de
        punten die als hoekpunten van grensvlakken gebruikt
        worden werkelijk in hetzelfde vlak liggen. Het min-
        teken in bijvoorbeeld P -Q R voorkomt dat D3D lijn-
        stuk PQ tekent. Omdat van iedere driehoek elke kant
        zichtbaar kan zijn, specificeren we de hoekpunten
        ervan zowel anti-kloksgewijs als kloksgewijs:
     */
     fprintf(fp2, "%d %d %d.\n", P, -(P+A+1), P+1);
     fprintf(fp2, "%d %d %d.\n", P+A+1, -P, P+A);
     fprintf(fp2, "%d %d %d.\n", P, -(P+A+1), P+A);
     fprintf(fp2, "%d %d %d.\n", P+A+1, -P, P+1);
   }
   fclose(fp2);
}
```

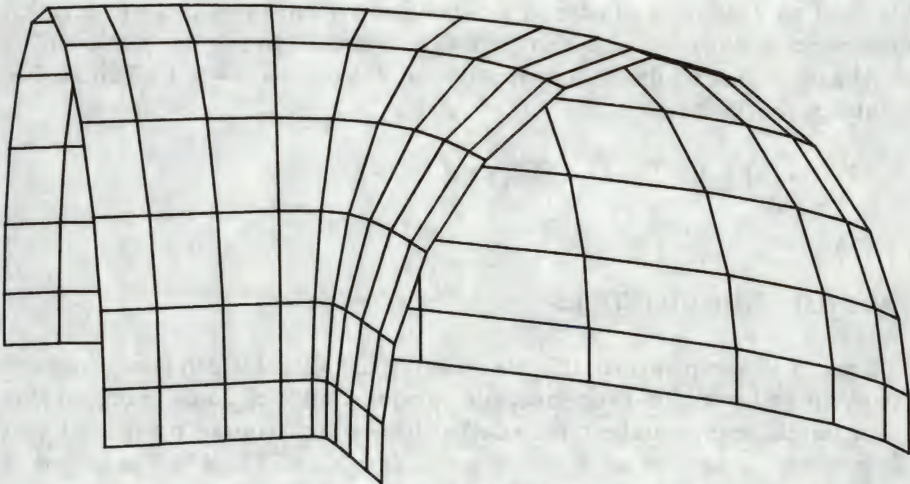
Figuur 3.19 is gemaakt door programma D3D, op basis van een file geproduceerd door programma BSPLSURF. De invoerfile voor laatstgenoemd programma bevat 42 punten, waarvan er sommige samenvallen. We hebben het idee om meer dan één nummer aan een punt toe te kennen besproken voor krommen, namelijk aan het eind van paragraaf 3.8; dit idee is hier toegepast op een oppervlak. In de volgende tabel is $n=6$ en $m=7$:

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	39	31	32	33	34	35
36	37	38	39	40	41	42

Iedere regel in deze tabel correspondeert met een gebroken lijn in figuur 3.18. De eerste regel betreft evenwel dezelfde punten als de tweede; evenzo vallen de punten van de vijfde regel samen met die van de zesde. Bovendien valt het eerste punt van iedere regel samen met het tweede en het zesde met het zevende. Als twee of meer punten samenvallen, laat D3D alleen het hoogste puntnummer van die punten zien. In figuur 3.18 geeft nummer 9 ook de punten 1, 2 en 8 aan. (Het idee om punten te laten samenvallen is beslist niet essentieel: als u dit alleen maar een vervelende complicatie vindt hoeft u er geen gebruik van te maken, maar u moet er wel steeds aan denken dat het eerste en het laatste punt van iedere kromme niet worden benaderd.)



Figuur 3.18 Invoerfile voor BSPLSURF afgebeeld met D3D



Figuur 3.19 B-spline oppervlak

Ook de puntnummers 16 en 17 ontbreken in figuur 3.18; zij vallen samen met de punten 23 en 24, maar dit heeft een heel andere reden. Ik heb figuur 3.18 geconstrueerd door het *Transform*-commando te gebruiken; de gebroken lijn gemerkt met de nummers 23, 24, 25, 26 en 28 werd verkregen door de gebroken lijn waarop punt 18 ligt over 90° te roteren om de as (23, 24), wat betekent dat de punten 15, 16 en 17 samenvallen met de punten 22, 23, 24. Door $N = M = 3$ te kiezen ontstond een file met 130 punten, die met behulp van D3D figuur 3.19 opleverde.
Het zojuist genoemde hoogste puntnummer, 130, biedt ons een goede gelegenheid de relatie te controleren die er is tussen enerzijds n, m, N, M, i, j en anderzijds het hoogste absolute puntnummer k dat in de uitvoerfile voorkomt. In het algemeen hebben we

$$k = \{(i - 2)N + I\}\mathrm{A} + (j - 2)M + 1 + J$$

waarin de waarde

$$A = (m - 3)M + 1$$

het aantal punten (in de uitvoerfile) aangeeft die liggen op één 'horizontale' lijn. In het voorbeeld van figuur 3.19 hebben we

$$m = 7, n = 6, M = 3, N = 3.$$
$$A = (7 - 3) \times 3 + 1 = 13$$

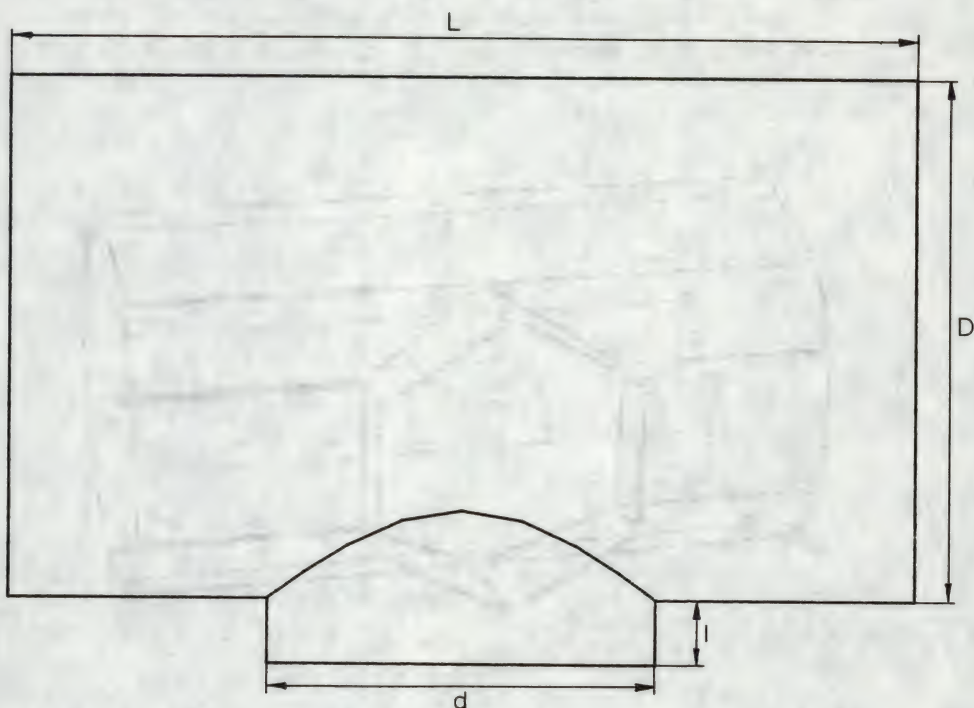
Omdat i en j gebruikt worden in for-statements waarin hun grootste waarden respectievelijk $n - 2$ en $m - 2$ zijn en de grootste waarden van I en J respectievelijk N en M zijn, vinden we de grootste waarde van k door $i = 4$, $j = 5$, $I = 3$ en $J = 3$ te gebruiken, wat leidt tot

$$\begin{aligned} k &= \{(4 - 2) \times 3 + 3\} \times 13 + (5 - 2) \times 3 + 1 + 3 \\ &= 130 \end{aligned}$$

3.11 SNIJDENDE CILINDERS

In figuur 1.13 van paragraaf 1.5 hebben we enkele afbeeldingen gezien van een voorwerp bestaande uit twee snijdende cilinders, ook wel *T-stuk* genoemd. We zullen nu een programma bespreken dat een file voor zo'n voorwerp genereert. Met dit programma kunnen we niet alleen de file EXAMPLE3.DAT, genoemd in paragraaf 1.3, genereren, maar ook files voor soortgelijke constructies met andere afmetingen. Bovendien kunnen we de benadering van cilinders verbeteren door het aantal grensvlakken te verhogen (uiteraard ten koste van meer rekentijd). Als u meer in het snijden van cilinders dan in het programmeren bent geïnteresseerd, dan kunt u het programma eenvoudigweg gebruiken, zonder u te bekommeren over de

programmadetails die in deze paragraaf worden besproken. Anderzijds kunt u deze paragraaf ook beschouwen als weer een andere demonstratie van het schrijven van programma's die objectfiles genereren.



Figuur 3.20 Bovenaanzicht van snijdende cilinders

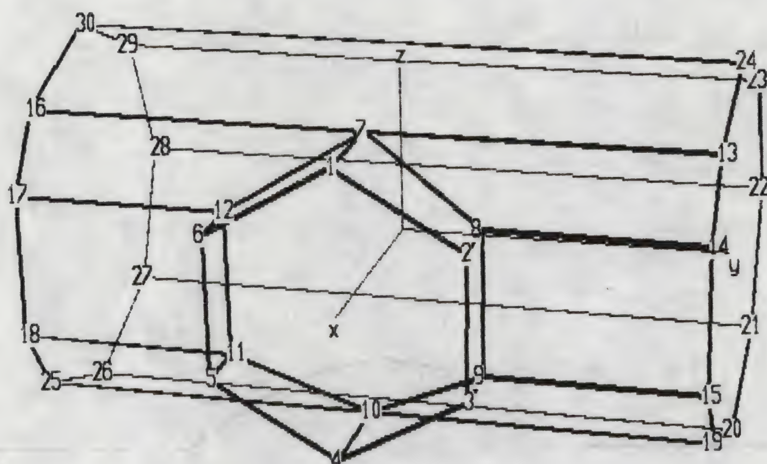
Figuur 3.20 is een bovenaanzicht van het voorwerp in kwestie. De volgende vier maten moeten door de gebruiker van het programma worden opgegeven:

- d, de diameter van de kleinste cilinder;
- D, de diameter van de grootste cilinder ($D > d$);
- l, de 'lengte' van de kleinste cilinder (zie figuur 3.20);
- L, de lengte van de grootste cilinder ($L > d$);

(De termen *klein* en *groot* moeten hier niet te letterlijk worden opgevat, want de lengte l van de 'kleinste' cilinder kan groter zijn dan de drie andere maten. We moeten deze begrippen dan ook eigenlijk niet op de lengte maar op de diameter

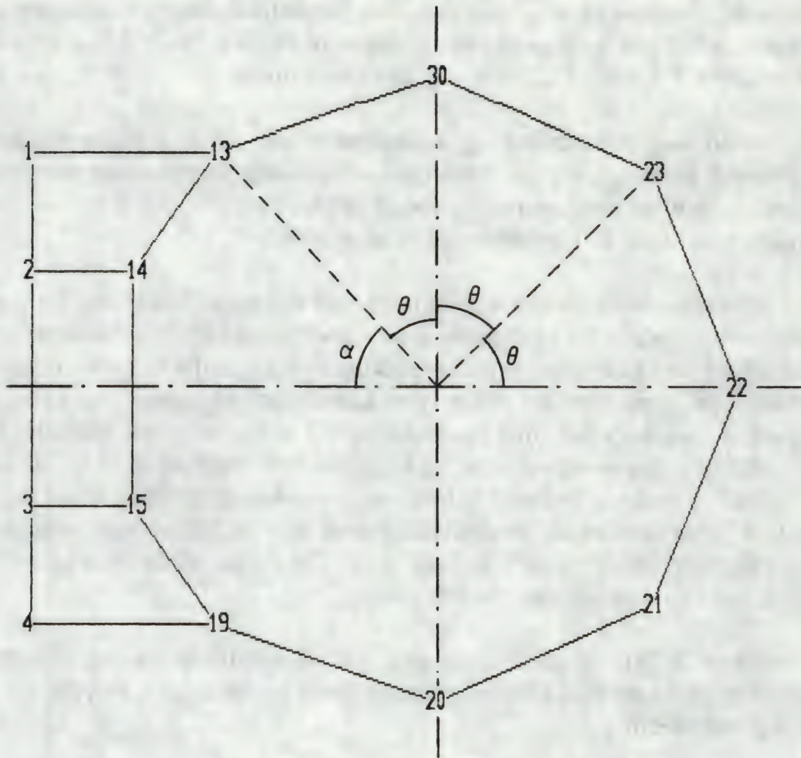
toepassen: de termen *dunne* en *dikke* cilinder zouden misschien beter op hun plaats zijn geweest.)

We zullen de oorsprong O van het coördinatenstelsel plaatsen in het snijpunt van de twee cilinderassen. De y-as zal samenvallen met de as van de grootste cilinder en de (positieve) x-as met die van de kleinste cilinder, zoals figuur 3.21 laat zien.



Figuur 3.21 Perspectivische afbeelding van snijdende cilinders (n = 3)

Zoals gebruikelijk, zullen we de cirkelvormige dwarsdoorsnede van de twee cilinders benaderen door middel van veelhoeken. Normaal gesproken zijn zulke veelhoeken *regelmatig*. Hier zal dat wel het geval zijn voor de kleinste cilinder maar niet voor de grootste. Dit komt omdat we willen dat de evenwijdige lijnen op de kleinste cilinder de overeenkomstige lijnen op de grootste cilinder snijden. Zo wordt bijvoorbeeld in figuur 3.21 het voorvlak van de kleinste cilinder benaderd door een regelmatige zeshoek. De lijn door hoekpunt 2 hiervan, evenwijdig aan de x-as, snijdt de grootste cilinder in punt 8 en lijnstuk 8-14 op de grootste cilinder bepaalt de ligging van punt 14 op het rechter zijvlak. Op die manier volgt in figuur 3.21 de positie van de punten 13, 14, 15 en 19 uit die van de punten 1, 2, 3 en 4 van de kleinste cilinder, waardoor de eerstgenoemde punten niet even ver uit elkaar liggen, zoals ook uit figuur 3.22 blijkt.



Figuur 3.22 Zijaanzicht van snijdende cilinders (n = 3)

De gebruiker van ons programma (TCYL) zal n moeten intypen, het aantal punten op de *helft* van de kleinste cirkel, dus in figuur 3.21 hebben we $n = 3$. (Uiteraard zal in de praktijk n groter dan 3 worden genomen.) Op de grootste cirkel kennen we de positie van punt 13, die hoek α bepaalt, zie figuur 3.22. We verdelen nu het supplement van α (dat wil zeggen $180° - \alpha$) in n gelijke hoeken, waardoor er in totaal $2n$ sectoren, elk met een hoek

$$\theta = \frac{180° - \alpha}{n}$$

en n sectoren van ongelijke grootte (voor de resterende hoek 2α) ontstaan.

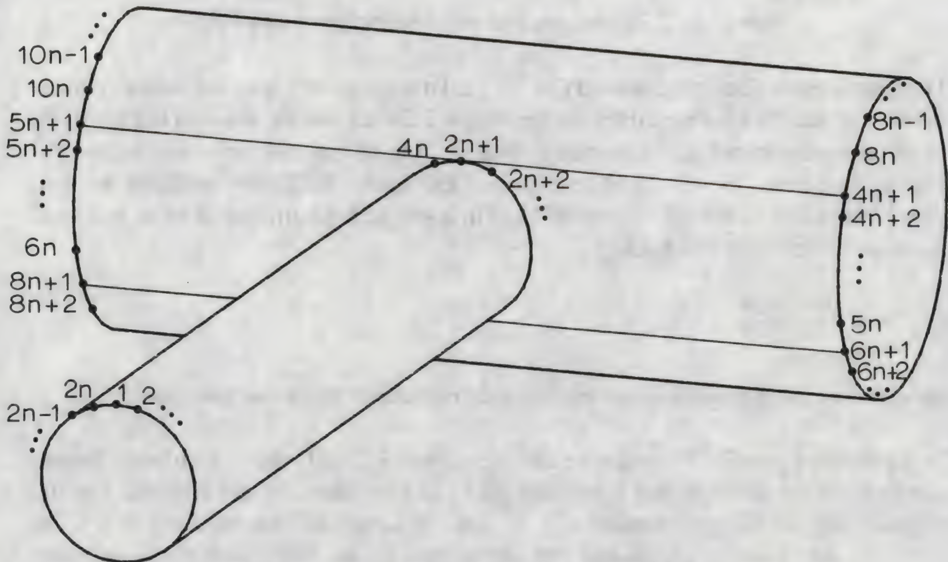
In plaats van figuur 3.22 kunnen we nu beter figuur 3.23 gebruiken, omdat de daarin aangegeven puntnummering algemeen geldig is. De punten op het voorvlak van de kleinste cilinder zijn genummerd 1, 2, ..., $2n$. De volgende $2n$ nummers ($2n + 1$, $2n + 2$, ..., $4n$) worden toegekend aan de punten op de snijkromme van de twee cilinders. Dan krijgen we de punten $4n + 1$, $4n + 2$, ..., $5n$, die liggen op het rechter zijvlak van de grootste cilinder. De overeenkomstige punten op het linker zijvlak

hebben de nummers $5n + 1, 5n + 2, ..., 6n$. Tenslotte kennen we nummers toe aan de resterende punten op deze twee zijvlakken: $6n + 1, 6n + 2, ..., 8n$ voor het rechter en $8n + 1, 8n + 2, ..., 10n$ voor het linker zijvlak.

Het vinden van de rechthoekige coördinaten van al deze hoekpunten is een betrekkelijk eenvoudige taak; moeilijker is het, correcte algemene uitdrukkingen te vinden voor de puntnummers, vooral omdat niet alle delen van de grootste cilinder op dezelfde manier behandeld kunnen worden.

Er is een probleempje dat te maken heeft met de mogelijkheid om 'kunstmatige' ribben weg te laten in het eindresultaat waarin verborgen lijnen onzichtbaar zijn, zie de figuren 1.13(b) en (c) en onze bespreking in paragraaf 1.5. Deze mogelijkheid werkt alleen goed als zo'n ribbe twee keer in de objectfile voorkomt en als bovendien twee keer *hetzelfde paar puntnummers* ervoor wordt gebruikt. Omdat bijvoorbeeld in figuur 3.21 lijnstuk 7-13 een zijde van veelhoek 8-14-13-7 is, kunnen we de twee lijnstukken 16-7 en 7-13 beter niet combineren in één zijde van rechthoek 16-13-24-30, maar in plaats daarvan moeten we deze rechthoek representeren door de (gedegenereerde) vijfhoek 16-7-13-24-30. Dit verklaart de twee grensvlakken met vijf hoekpuntnummers in de objectfile.

Programma TCYL bevat commentaar dat aangeeft met welk deel van het oppervlak we bezig zijn. Het programma dient in combinatie met figuur 3.23 te worden bestudeerd.



Figuur 3.23 Puntnummering in snijdende cilinders

```
/* TCYL: Programma voor het maken van een D3D-objectfile
         voor een T-vormig voorwerp, bestaande uit
         twee snijdende cilinders.
*/
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <alloc.h>

void wri(int a);
void wr4(int a, int b, int c, int d);
void endcode(void);

FILE *fp;

main()
{ char filnam[30];
  int i, n, n2, n3, n4, n5, n6, n8, n9, n10,
      tablelength;
  float a, b, l, L, D, d, R, r, alpha, theta, delta,
      *x, *y, *z, pi, beta;
  pi = 4.0 * atan(1.0);
  printf(
"\n------------------------ lengte L, diameter D\n");
  printf("           |\n");
  printf("           |\n");
  printf("           | lengte l, diameter d\n");
  printf("           |\n");
  printf("           |\n\n");

  do
  { printf("\n(d < D,  d < L)\n\n");
    printf("Geef d, D, l, L: \n");
    scanf("%f %f %f %f", &d, &D, &l, &L);
  } while (d >= D || d >= L);
  R = D/2; r = d/2; a = l + R; b = L/2;
  printf(
"\nHet cirkelvormig grensvlak met diameter d zal worden\n");
"benaderd door een regelmatige veelhoek met 2n hoeken.\n\n");
  printf("Geef n: "); scanf("%d", &n);
  printf("Naam van de uitvoerfile: "); scanf("%s", filnam);
  fp = fopen(filnam, "w");
  if (fp == NULL) {printf("File-probleem"); exit(1);}
  n2 = 2*n; n3 = 3*n; n4 = 4*n; n5 = 5*n;
```

```
n6 = 6*n; n8 = 8*n; n9 = 9*n; n10 = 10*n;
tablelength = (n10 + 1) * sizeof(float);
x = (float *)malloc(tablelength);
y = (float *)malloc(tablelength);
z = (float *)malloc(tablelength);
if (z == NULL) {printf("Niet genoeg geheugen"); exit(1);}
theta = pi/n;
for (i=1; i<=n2; i++)
{ x[i] = a;
  y[i] = r*sin((i-1)*theta);
  z[i] = r*cos((i-1)*theta);
}
for (i=n2+1; i<=n4; i++)
{ y[i] = y[i-n2]; z[i] = z[i-n2];
  x[i] = sqrt(R*R - z[i]*z[i]);
}
for (i=n4+1; i<=n5; i++)
{ x[i] = x[i-n2]; y[i] = b; z[i] = z[i-n2];
}
for (i=n5+1; i<=n6; i++)
{ x[i] = x[i-n]; y[i] = -b; z[i] = z[i-n];
}
alpha = atan(z[n4+1]/x[n4+1]);
delta = (pi - alpha)/n;
for (i=n6+1; i<=n8; i++)
{ beta = alpha + (i-n6-1)*delta;
  x[i] = R * cos(beta); y[i] = b; z[i] = -R * sin(beta);
}
for (i=n8+1; i<=n10; i++)
{ x[i] = x[i-n2]; y[i] = -b; z[i] = z[i-n2];
}
for (i=1; i<=n10; i++)
  fprintf(fp, "%d %f %f %f\n", i, x[i], y[i], z[i]);

fprintf(fp, "\nFaces:\n");

/* Kleinste cilinder, gekromd oppervlak: */
for (i=1; i<n2; i++) wr4(i, i+1, i+n2+1, i+n2);
wr4(n2, 1, n2+1, n4);

/* Grootste cilinder, stukken op gekromd oppervlak rechts
   van kleinste cilinder; van boven naar beneden:
*/
for (i=n2+1; i<n3; i++) wr4(i, i+1, i+n2+1, i+n2);
wr4(n3, n3+1, n6+1, n5);
```

```
/* Grootste cilinder, stukken op gekromd oppervlak links
   van kleinste cilinder; van beneden naar boven:
*/
wr4(n3+2, n6, n8+1, n3+1);
for (i=n3+3; i<=n4; i++) wr4(i, n9+2-i, n9+3-i, i-1);
wr4(n2+1, n5+1, n5+2, n4);

/* Grootste cilinder, grote rechthoekige stukken op
   gekromd oppervlak, beginnend bij het onderste punt
   van de kleinste cilinder:
   */
   wri(n6+2); wr4(n6+1, n3+1, n8+1, n8+2);
   for (i=n6+3; i<=n8; i++) wr4(i, i-1, i+n2-1, i+n2);
   wri(n4+1); wr4(n8, n10, n5+1, n2+1);

   /* Voorvlak van kleinste cilinder: */
   for (i=n2; i>0; i--) wri(i);
   endcode();

   /* Rechter zijvlak van grootste cilinder:            */
   for (i=n4+1; i<=n5; i++) wri(i); /* Voorste deel     */
   for (i=n6+1; i<=n8; i++) wri(i); /* Rest             */
   endcode();

   /* Linker zijvlak van grote cilinder:                */
   for (i=n6; i>n5; i--) wri(i);    /* Voorste deel     */
   for (i=n10; i>n8; i--) wri(i);   /* Rest             */
   endcode();
   fclose(fp);
}

void wri(int a)
{ fprintf(fp, " %d", a);
}

void wr4(int a, int b, int c, int d)
{ fprintf(fp, " %d %d %d %d.\n", a, b, c, d);
}

void endcode(void)
{ fprintf(fp,".\n");
}
```

3.12 EEN PROGRAMMA VOOR VIERKANT SCHROEFDRAAD

We zullen ons nu gaan bezighouden met programma SCRTHR, dat we hebben gebruikt in paragraaf 2.6. Zoals bekend, produceert het een objectfile voor vierkant schroefdraad, afgebeeld in figuur 2.21. We zullen de bespreking van paragraaf 2.6 hier niet gaan herhalen, maar liever onze aandacht richten op de puntnummering en de benadering van gekromde oppervlakken door platte grensvlakken.

Laten we beginnen met de terminologie die we zullen gebruiken. Zoals u zich herinnert zijn een grote diameter D en een kleine diameter d gegeven; in het programma is het (net als in de wiskunde) prettiger met een straal dan met een diameter te werken, dus we definiëren

$$R = D/2, r = d/2.$$

De as van de schroefdraad zal samenvallen met de positieve z-as, wat inhoudt dat de schroef verticaal is geplaatst.

We onderscheiden:

1 Het *buitenoppervlak*, waarvan alle punten een afstand R van de as verwijderd zijn.

2 Het *binnenoppervlak*, waarvan alle punten een afstand r van de as verwijderd zijn.

3 Het *bovenvlak*, een plat grensvlak liggend in het vlak $z = L$.

4 Het *grondvlak*, een plat grensvlak liggend in het xy-vlak ($z = 0$).

5 De *bovenflank*, een gekromd oppervlak, gedeeltelijk zichtbaar in figuur 3.24, dat het binnenoppervlak verbindt met het buitenoppervlak.

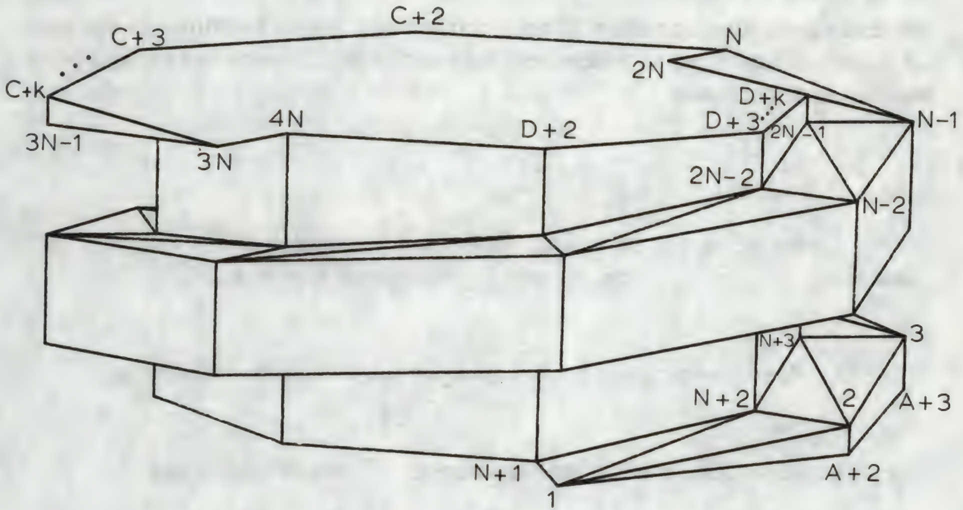
6 De *onderflank*, een gekromd oppervlak, analoog aan de bovenflank en gedeeltelijk zichtbaar in figuur 3.25.

Ons schroefdraad bevat gekromde niet-horizontale randen; zo'n kromme wordt *helix* genoemd. Ten behoeve van de benaderingsmethode moet de gebruiker k opgeven, het aantal stappen in een halve omwenteling. Door uit te gaan van een halve omwenteling zijn we er zeker van dat

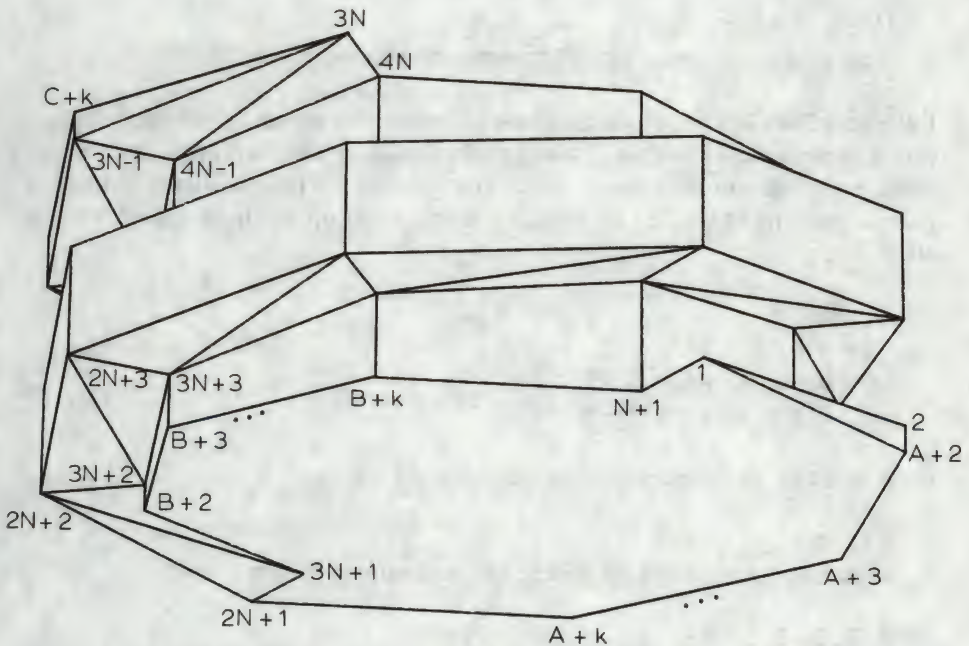
$$n = 2k,$$

het aantal stappen in een hele omwenteling, een even getal is. Omdat de gegeven

steek (*pitch*) gelijk is aan de voorwaartse verplaatsing van de schroef in een volledige omwenteling, dat wil zeggen in *n* stappen, neemt de z-coördinaat toe met



Figuur 3.24 Schroefdraad, benaderd door platte grensvlakken; schuin van boven bekeken



Figuur 3.25 Schroefdraad, benaderd door platte grensvlakken; schuin van onderen bekeken

$$delta\ L = \frac{pitch}{n}$$

als we een helix één stap volgen. De gegeven gewenste lengte L_0 wordt dan afgerond op L, het dichtstbijzijnde gehele veelvoud van *delta L*. We zullen vaak gebruik maken van de constante

$$N = \frac{L}{delta\ L} + 1,$$

die het aantal gebruikte punten op iedere helix aangeeft. We nummeren deze punten, van onder naar boven, als volgt (zie de figuren 3.24 en 3.25):

$1, 2, \ldots, N$:
Helix 1, de snijkromme van het buitenoppervlak met de bovenflank;

$N+1, N+2, \ldots, 2N$:
Helix 2, de snijkromme van het binnenoppervlak met de bovenflank;

$2N+1, 2N+2, \ldots, 3N$:
Helix 3, de snijkromme van het buitenoppervlak met de onderflank;

$3N+1, 3N+2, \ldots, 4N$:
Helix 4, de snijkromme van het binnenoppervlak met de onderflank;

Behalve de vier helixen moeten we ook het bovenvlak en het grondvlak bekijken; op elk hiervan vinden we twee halve cirkels, de ene met straal R en de andere met straal r. Op elk van deze vier halve cirkels liggen k - 1 punten waaraan nog een puntnummer moet worden toegekend omdat zij niet op een helix liggen. Met de afkortingen

$A = 4N - 1,$
$B = A + k - 1 = 4N + \mathrm{k} - 2,$
$C = B + k - 1 = 4N + 2k - 3,$
$D = C + k - 1 = 4N + 3k - 4,$

doen we dit als aangegeven in de figuren 3.24 en 3.25:

$A+2, A+3, \ldots, A+k$:
Resterende punten op snijkromme van buitenoppervlak en grondvlak;

$B+2, B+3, \ldots, B+k$:
Resterende punten op snijkromme van binnenoppervlak en grondvlak;

C+2, C+3, ..., C+k:
Resterende punten op snijkromme van buitenoppervlak en bovenvlak;

D+2, D+3, ..., D+k:
Resterende punten op snijkromme van binnenoppervlak en bovenvlak.

Onderaan beginnend, moeten de eerste en de laatste 180° van het buitenoppervlak enigszins anders behandeld worden dan de rest (dat we het *middengedeelte* zullen noemen) en hetzelfde is van toepassing op het binnenoppervlak. We benaderen het middengedeelte van het buiten- en binnenoppervlak door middel van parallellogrammen met twee verticale zijden. Het allereerste stuk van het buitenoppervlak is een driehoek met puntnummers 1, A+2, 2. Daarna komen k - 1 trapezia: deze hebben twee verticale zijden van ongelijke lengte. Iets dergelijks geldt voor de laatste 180°; het allerlaatste stuk van het buitenoppervlak is de driehoek met hoekpunten 3N, C+k, 3N-1. We krijgen een vergelijkbare situatie voor het binnenoppervlak, dat begint met de driehoek B+2, 3N+2, 3N+1 en eindigt met de driehoek D+k, 2N-1, 2N.

Het zal u misschien verwonderen dat de boven- en onderflank in de figuren 3.24 en 3.25 een verdeling in driehoeken laat zien: het gebruik van vierhoeken lijkt hier eenvoudiger en natuurlijker. Om de waarheid te zeggen, ik maakte dezelfde vergissing in de allereerste versie van mijn programma. Een poging om bijvoorbeeld het oppervlakdeel 1, 2, N+2, N+1 (zie figuur 3.24) als één grensvlak op te geven levert een foutmelding van D3D op, waarbij geklaagd wordt dat niet alle hoekpunten van het grensvlak in hetzelfde vlak liggen. We kunnen dit als volgt verklaren. De hoekpunten 1 en N+1 hebben dezelfde z-coördinaat (z=0) en hetzelfde geldt voor de punten 2 en N+2 (z=*pitch*/*n*). De lijnen (1, N+1) en (2, N+2) liggen daarom in twee verschillende horizontale vlakken. Dit betekent dat deze twee horizontale lijnen evenwijdig zouden zijn als de vier genoemde punten in één vlak lagen. (Immers, elk niet-horizontaal vlak snijdt twee horizontale vlakken volgens twee evenwijdige lijnen.) Het zal duidelijk zijn dat de twee horizontale lijnen in kwestie niet evenwijdig zijn, dus de vier genoemde punten liggen niet in hetzelfde vlak. Hiermee is ook aangetoond dat, in tegenstelling tot wat u misschien had verwacht, de twee lijnstukken (1, 2) en (N+1, N+2) niet evenwijdig zijn (waarbij de helling van eerstgenoemd lijnstuk minder steil is dan die van het tweede).

Het grond- en bovenvlak zijn niet-convexe veelhoeken, die we elk als één grensvlak kunnen specificeren. Afgezien van enige discontinuïteiten in hun hoekpuntnummers zijn zij niet moeilijk, vooral niet als u gebruik kunt maken van de figuren 3.24 en 3.25. Uiteraard had ik deze illustraties nog niet toen ik begon na te denken over het schrijven van programma SCRTHR en evenmin beschikte ik over een echt model of over een foto van een stuk vierkant schroefdraad met een plat eindvlak, zodat het aardig wat tijd kostte om zo'n eindvlak te schetsen. Met bovengenoemde puntnummering is programma SCRTHR betrekkelijk eenvoudig. Het is misschien

instructief te vermelden dat ik eerst een heel andere puntnummering gebruikte en een programma schreef dat zo ingewikkeld was dat ik het zelf niet meer begreep toen ik het nodig had voor dit boek. Toch werkte het programma correct, zodat ik er enkele plaatjes zoals figuur 3.24 mee kon maken. Hiervan gebruik makend, stapte ik over op de boven besproken puntnummering en schreef een volkomen nieuw programma, SCRTHR, dat hieronder is opgenomen.

```
/* SCRTHR:  Screw-thread
   */
   #include <stdio.h>
   #include <math.h>
   #include <ctype.h>
   #include <alloc.h>
   #include <process.h>

   struct point {float x, y, z;} *p;
   FILE *fp;

   void p1(int i);
   void p1e(int i);
   void p3e(int i, int j, int k);
   void p4e(int i, int j, int k, int l);
   void pr(int a, int b);

  main()
  { int i, j, n, N, N2, N3, N4, k, steps, A, B, C, D;
    double pi, theta, phi, R, r, deltaL,
       major, minor, pitch, h, L, L0, offset;
    char str[30], ch;
    pi = 4 * atan(1.0);
    printf("Grote diameter D:  "); scanf("%lf", &major);
    printf("Kleine diameter d: "); scanf("%lf", &minor);
    printf(
    "Bij echt vierkant draad hebben we:\n");
    printf("\n   pitch = D - d = %5.3f\n\n",
    pitch = major - minor);
    printf(
    "In een volledige omwenteling is 'pitch' de afstand\n");
    printf(
    "die de schroef voorwaarts beweegt, dus het 'vierkant'\n");
    printf("heeft zijden met lengte\n");
    printf("\n   h = pitch/2 = %5.3f\n\n", pitch/2);
    printf(
    "Wilt u de steek (pitch) anders kiezen, zodat de draad\n");
```

```
printf(
"niet echt vierkant (maar rechthoekig) is? (J/N): ");
scanf("%s", str);
ch = toupper(str[0]);
if (ch == 'J')
{ printf("Pitch: "); scanf("%lf", &pitch);
}
h = pitch/2;
printf("Gewenste lengte van de schroefdraad\n");
printf("(niet kleiner dan %5.3f): ", h);
scanf("%lf", &L0);
if (L0 < h) L0 = h; /* Ten minste een halve omwenteling */
printf("Aantal stappen in een halve omwenteling: ");
scanf("%d", &k);
n = 2 * k;
deltaL = pitch/n;
steps = (int)(L0/deltaL + 0.5); N = steps + 1;
N2 = 2*N; N3 = 3*N; N4 = 4*N;
L = steps * deltaL;
printf("De werkelijke lengte wordt: %5.3f\n", L);
printf("Naam van uitvoerfile: "); scanf("%s", str);
fp = fopen(str, "w");
theta = pi/k;
R = major/2; r = minor/2;
A = 4 * N - 1;
B = A + k - 1;
C = B + k - 1;
D = C + k - 1;
p = (struct point *)malloc((D+k+1)*sizeof(struct point));
if (p == NULL) {printf("Niet genoeg geheugen"); exit(1);}
for (i=1; i<=N; i++)       /* Buitenste helix op bovenflank */
{ phi = (i-1)*theta;
  p[i].x = R * cos(phi);
  p[i].y = R * sin(phi);
  p[i].z = (i-1) * deltaL;
}
for (i=N+1; i<=N2; i++)   /* Binnenste helix op bovenflank */
{ p[i].x = p[i-N].x * r/R;
  p[i].y = p[i-N].y * r/R;
  p[i].z = p[i-N].z;
}
for (i=N2+1; i<=N3; i++)  /* Buitenste helix op onderflank */
{ p[i].x = -p[i-N2].x;
  p[i].y = -p[i-N2].y;
```

```
  p[i].z = p[i-N2].z;
}
for (i=N3+1; i<=N4; i++)  /* Binnenste helix op onderflank */
{ p[i].x = p[i-N].x * r/R;
  p[i].y = p[i-N].y * r/R;
  p[i].z = p[i-N].z;
}
for (i=A+2; i<=A+k; i++)
                    /* Buitenste halve cirkel in grondvlak */
{ phi = (i-A-1)*theta;
  p[i].x = R * cos(phi);
  p[i].y = R * sin(phi);
  p[i].z = 0;
}
for (i=B+2; i<=B+k; i++)
                    /* Binnenste halve cirkel in grondvlak */
{ p[i].x = -p[i-B+A].x * r/R;
  p[i].y = -p[i-B+A].y * r/R;
  p[i].z = 0;
}
offset = steps * theta;
while (offset > 2*pi) offset -= 2*pi;
for (i=C+2; i<=C+k; i++)
                    /* Buitenste halve cirkel in bovenvlak */
{ phi = offset + (i-C-1)*theta;
  p[i].x = R * cos(phi);
  p[i].y = R * sin(phi);
  p[i].z = L;
}
for (i=D+2; i<=D+k; i++)
                    /* Binnenste halve cirkel in bovenvlak */
{ p[i].x = -p[i-D+C].x * r/R;
  p[i].y = -p[i-D+C].y * r/R;
  p[i].z = L;
}
for (i=1; i<=D+k; i++)
  fprintf(fp, "%4d %f %f %f\n",
             i, p[i].x, p[i].y, p[i].z);

fprintf(fp, "Faces:\n");

/* z = 0: */
p1(N2+1); pr(A+k, A+2); p1(1); p1(N+1);
pr(B+k, B+2); p1e(N3+1);
```

```
/* z = L: */
p1(N); pr(C+2, C+k); p1(N3); p1(N4);
pr(D+2, D+k); p1e(N2);

/* Eerste 180 graden (onderaan) */

/* Buitenoppervlak: */
p3e(1, A+2, 2);
for (j=2; j<k; j++) p4e(A+j, A+j+1, j+1, j);
p4e(A+k, N2+1, k+1, k);

/* Binnenoppervlak: */
p3e(B+2, N3+2, N3+1);
for (j=2; j<k; j++) p4e(B+j+1, N3+j+1, N3+j, B+j);
p4e(N+1, N3+k+1, N3+k, B+k);

/* Laatste 180 graden (bovenaan) */

/* Buitenoppervlak: */
p3e(N3, C+k, N3-1);
for (j=2; j<k; j++) p4e(C+k-j+2, C+k-j+1, N3-j, N3-j+1);
p4e(C+2, N, N3-k, N3-k+1);

/* Binnenoppervlak: */
p3e(D+k, N2-1, N2);
for (j=2; j<k; j++) p4e(D+k-j+2, D+k-j+1, N2-j, N2-j+1);
p4e(D+2, N4, N2-k, N2-k+1);

/* Middengedeelte */

/* Buitenoppervlak: */
for (j=1; j<N-k; j++) p4e(N2+j, N2+j+1, k+j+1, k+j);

/* Binnenoppervlak: */
for (j=1; j<N-k; j++) p4e(N+j, N+j+1, N3+k+j+1, N3+k+j);

/* Bovenflank: */
for (j=1; j<N; j++)
{ p3e(j, N+j+1, N+j);
  p3e(N+j+1, j, j+1);
}

/* Onderflank: */
for (j=1; j<N; j++)
{ p3e(N3-j+1, N4-j, N4-j+1);
```

```
p3e(N4-j, N3-j+1, N3-j);
  }
  fclose (fp);
}

void p1(int i)
{ fprintf(fp, " %d", i);
}

void p1e(int i)
{ fprintf(fp, " %d.\n", i);
}

void p3e(int i, int j, int k)
{ fprintf(fp, " %d %d %d.\n", i, j, k);
}

void p4e(int i, int j, int k, int l)
{ fprintf(fp, " %d %d %d %d.\n", i, j, k, l);
}

void pr(int a, int b)
{ int i;
  if (b >= a) for (i=a; i<=b; i++) fprintf(fp, " %d", i);
        else for (i=a; i>=b; i--) fprintf(fp, " %d", i);
}
```

4 DE PROGRAMMA'S D3D EN PLOTHP

4.1 INLEIDING

Dit hoofdstuk gaat voornamelijk over programma D3D, dat groter is dan alle andere programma's die we tot dusverre hebben besproken. De uitvoerbare versie, D3D.EXE, wordt verkregen door de vier objectmodules D3D.OBJ, GRPACK.OBJ, HLPFUN.OBJ en TRAFO.OBJ te 'linken'. Deze objectmodules zijn op hun beurt geproduceerd door de Turbo C compiler bij het compileren van de overeenkomstige vier files (D3D.C, GRPACK.C, HLPFUN.C, TRAFO.C) met programmatekst. Hieronder is aangegeven waar u een korte bespreking van deze vier modulen en de programmatekst zelf kunt vinden:

Module	Bespreking	Programmatekst
D3D	Paragraaf 4.2	Appendix A
GRPACK	Paragraaf 4.3	Paragraaf 4.3
HLPFUN	Paragraaf 4.4	Appendix B
TRAFO	Paragraaf 3.7	Paragraaf 3.7

Zoals u zich zult herinneren, hebben we in paragraaf 3.7 driedimensionale rotaties besproken in verband met krommen en kabels; dezelfde functies (*initrotate* en *rotate*) die we daar hebben gebruikt zullen ook hier nuttig blijken te zijn.

Programma D3D is te groot om in dit boek in detail uitgelegd te worden, dus we zullen er slechts enkele aspecten van bespreken. Als u er meer van wilt weten dan in paragraaf 4.2 wordt behandeld, dan zou u zich kunnen wagen aan het bestuderen van de programmatekst zelf, opgenomen in Appendix A.

Paragraaf 4.3 gaat over GRPACK, een pakket van grafische routines dat niet alleen in verband met D3D maar ook voor tal van andere toepassingen nuttige diensten kan bewijzen. Het lijkt veel op een grafisch pakket met dezelfde naam, opgenomen en besproken in mijn boek *Computer Graphics for the IBM PC*. Dat boek was gebaseerd op de IBM Color Graphics Adapter (CGA) en de Hercules Graphics Adapter (HGA), terwijl de hier opgenomen versie van GRPACK bovendien het gebruik van de Enhanced Graphics Adapter (EGA) ondersteunt. Verder is de

nieuwe versie van GRPACK gebaseerd op Turbo C, een compiler die nog niet beschikbaar was toen ik bovengenoemd boek schreef. Nog een nieuw aspect is de mogelijkheid om grafische uitvoer in een file te verkrijgen, die later gelezen kan worden door een ander programma om zodoende een 'hard copy' te produceren, bijvoorbeeld met een plotter. Een voorbeeld van zo'n programma wordt gegeven in paragraaf 4.5; we zullen ook zo dadelijk nog even hierop terugkomen.

In paragraaf 4.4 zullen we HLPFUN bespreken, een functie voor het *weglaten van verborgen lijnen*. De gebruikte methode is in principe dezelfde als die welke gebruikt is in HIDLINPIX, een programma dat een centrale rol speelt in mijn boek *Programming Principles in Computer Graphics*. Als uitbreiding op dat programma biedt HLPFUN faciliteiten voor het werken met gekromde oppervlakken; er zijn ook enkele andere verschilpunten met het oorspronkelijke programma. Zo wordt bijvoorbeeld de geheugenruimte op een meer flexibele manier gebruikt; ook is het programma nu niet meer recursief, zodat een eventueel gevaar van 'stack overflow' vermeden wordt.

Na de bespreking van de modulen die tezamen programma D3D vormen, volgt paragraaf 4.5, dat gaat over een zelfstandig programma, PLOTHP. Dit kan tekeningen op een HP-plotter produceren; op deze manier zijn vele illustraties in boek gemaakt. Zelfs in het geval dat u niet over zo'n plotter beschikt kunt u misschien nuttig gebruik van dit programma maken, omdat het ook kan dienen om grafische uitvoer naar het beeldscherm en naar een matrix-printer te sturen. Bovendien kunnen plotfiles, die we verkrijgen worden door (na de commando's *H* of *E*) de letter *O* te typen, ook worden gelezen door programma's die u zelf heeft geschreven; op deze manier kunt in principe met zulke zelf gemaakte programma's de uitvoer van D3D ook krijgen op andere grafische apparatuur, zoals bijvoorbeeld een laserprinter, met andere woorden, u kunt programma PLOTHP ook beschouwen als alleen maar een voorbeeld van een programma dat pas na de uitvoering van D3D het grafische eindprodukt produceert.

4.2 D3D-HOOFDMODULE

Voordat we ons met programma D3D bezighouden zullen we eerst eens bekijken hoe we door berekening perspectivische beelden van driedimensionale voorwerpen kunnen verkrijgen. We zullen gebruik maken van een *centraal objectpunt* O, dat ligt binnen het voorwerp dat we perspectivisch willen afbeelden. Een ander belangrijk punt is het oogpunt E, gegeven door de bolcoördinaten ρ, θ en φ, zoals we besproken hebben in paragraaf 1.2. Deze bolcoördinaten hebben betrekking op een rechts coördinatenstelsel, dat verkregen wordt door een zodanige translatie van het *wereldcoördinatenstelsel* van de gebruiker, dat genoemd punt O de oorsprong wordt. Dus ρ is gelijk aan EO. Voor ieder hoekpunt van het af te beelden voorwerp zijn de x_w, y_w en z_w, dat wil zeggen de wereldcoördinaten, gegeven; de oorsprong

van het wereldcoördinatenstelsel kan dus verschillen van O. De z-as ervan wijst omhoog en in de regel wijst de x-as naar ons toe.

We gebruiken de wereldcoördinaten x_w, y_w, z_w om de *oogcoördinaten* x_e, y_e, z_e te berekenen. Het oogcoördinatenstelsel is een links stelsel. De oorsprong ervan ligt in oogpunt E, de positieve z-as wijst naar het centrale objectpunt O, de x-as wijst horizontaal naar rechts en de y-as (loodrecht op de x- en op de z-as) wijst omhoog. De oogcoördinaten zijn het resultaat van de zogenaamde *kijktransformatie* ('viewing transformation'):

$$[x_e\ y_e\ z_e\ 1] = [x_w - x_O \quad y_w - y_O \quad z_w - z_O \quad 1]\, V$$

In deze matrixvermenigvuldiging gebruiken we de 4 x 4 matrix

$$V = \begin{bmatrix} -\sin\theta & -\cos\varphi\cos\theta & -\sin\varphi\cos\theta & 0 \\ \cos\theta & -\cos\varphi\sin\theta & -\sin\varphi\sin\theta & 0 \\ 0 & \sin\varphi & -\cos\varphi & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix}$$

Let erop dat we in bovenstaand matrixprodukt de termen $-x_O$, $-y_O$, $-z_O$ zouden kunnen weglaten als ze gelijk aan nul zijn, dat wil zeggen als het wereldcoördinatenstelsel O als oorsprong heeft.

De kijktransformatie is slechts de eerste van twee stappen die we moeten nemen. We gebruiken de oogcoördinaten om de tweedimensionale *schermcoördinaten* x_s en y_s te berekenen, die we nodig hebben voor het gewenste perspectivische beeld. We verkrijgen ze in de tweede stap, de *perspectivische transformatie*:

$$x_s = d \,.\, x_e + c_1$$
$$y_s = d \,.\, y_e + c_2$$

Het zal duidelijk zijn dat d een schaalfactor is en dat c_1 en c_2 gebruikt worden om het beeld in horizontale en verticale richting naar de juiste positie te verplaatsen. Mijn boek *Programming Principles in Computer Graphics* (*PPCG*) laat zien hoe matrix V kan worden gevonden als produkt van enkele andere matrices. Het bevat ook een paragraaf over het automatisch aanpassen van de grootte en positie aan een gegeven *viewport* (dat is het gedeelte van het scherm dat gebruikt moet worden). Ook dit hebben we nu nodig. Bij het gebruik van D3D kunnen punten worden ingevoerd waarvan het beeld buiten de viewport valt. In dat geval moeten we overgaan op andere waarden van c_1, c_2 en d, die zo moeten worden gekozen dat het gehele beeld weer binnen de viewport valt. Niet alleen de wereldcoördinaten x_w, y_w en z_w, maar ook de oogcoördinaten x_e, y_e en z_e zullen intern worden opgeborgen. Dit reduceert

de hoeveelheid werk bij het aanpassen van de beeldgrootte aanzienlijk, omdat we nu bij nieuwe waarden van c_1, c_2 en d alleen maar de perspectivische transformatie hoeven uit te voeren. Bedenk wel dat dit werk op alle hoekpunten van toepassing is, dus het zou een aanzienlijke verspilling van tijd zijn als we in dat geval de meer tijdrovende kijktransformatie onnodig zouden uitvoeren.

In (bovengenoemd boek) *PPCG* wijken de invoerfiles voor het daar besproken programma HIDLINPIX iets af van onze objectfiles: de coördinaten van alle hoekpunten worden daar voorafgegaan door de coördinaten x_O, y_O, z_O van het centrale objectpunt O. De situatie is nu iets anders in de zin dat we met een interactief programma bezig zijn, dat op elk moment nieuwe punten accepteert en niet van de gebruiker verlangt dat hij van te voren een centraal objectpunt opgeeft. Bij een gegeven verzameling punten, die hetzij uit een objectfile worden gelezen hetzij met de hand worden ingevoerd, zullen we daarom het centrale objectpunt berekenen en wel als volgt:

$$x_O = (xwmin + xwmax)/2$$
$$y_O = (ywmin + ywmax)/2$$
$$z_O = (zwmin + zwmax)/2,$$

waarin *xwmin* en *xwmax* de kleinste en de grootste x-(wereld-)coördinaten zijn, enzovoort. Dus punt O ligt precies in het midden van het kleinste rechthoekige prisma (met grensvlakken evenwijdig aan de assen van het wereldcoördinatenstelsel) waarin alle punten gelegen zijn. Laten we zo'n prisma een *begrenzend prisma* noemen. Als een nieuw punt wordt ingevoerd dat buiten het huidige begrenzende prisma ligt, dan wordt het begrenzende prisma vergroot; strikt genomen zouden we dan ook een nieuw centraal objectpunt moeten berekenen. Maar dan zouden ook de oogcoördinaten x_e, y_e en z_e bijgewerkt moeten worden, aangezien punt O immers van invloed is op het oogcoördinatenstelsel. We weten echter dat de positie van punt O niet erg kritisch is zo lang het niet al te ver van het centrum van het voorwerp af ligt. We laten daarom punt O onveranderd zo lang het ligt binnen een redelijke afstand van het centrum van het begrenzende prisma. Wanneer deze afstand, die we beschouwen als een 'tolerantie', wordt overschreden, dan moeten we toch echt punt O bijwerken, waarbij we accepteren dat het berekenen van nieuwe oogcoördinaten voor alle hoekpunten nogal wat tijd kan kosten. In programma D3D worden de volgende tolerantiewaarden voor de x-, y- en z-richting gebruikt:

$$xtol = 0.4(xwmax - xwmin)$$
$$ytol = 0.4(ywmax - ywmin)$$
$$ztol = 0.4(zwmax - zwmin)$$

Dit betekent dat we ons binnen het begrenzende prisma een gelijkvormig concentrisch prisma moeten voorstellen, verkregen door schaling van het

begrenzende prisma met een factor 0.8. Zo lang punt O binnen dit kleinste prisma blijft, beschouwen we het acceptabel.

Het hart van programma D3D is een while-lus, waarin elke keer vanaf het toetsenbord de eerste letter van een commando wordt gelezen, waarna de door dat commando voorgeschreven actie meteen wordt uitgevoerd. In de meeste gevallen gebeurt dit laatste door het aanroepen van een functie. Als de gebruiker bijvoorbeeld de letter *F* (of *f*) intypt, dan wordt de functie *faces* aangeroepen. In de module D3D zijn alle functies in alfabetische volgorde gerangschikt, wat het opzoeken ervan vergemakkelijkt. Alle hoekpunten worden in een tabel opgeborgen; in plaats van een gewoon array wordt een pointervariabele *p* gebruikt, wat ons in staat stelt een tabellengte te gebruiken die afhangt van de hoeveelheid geheugen die beschikbaar is. Voor elk hoekpunt worden, zoals eerder besproken, de wereldcoördinaten (x_w, y_w, z_w) en de oogcoördinaten (x_e, y_e, z_e) opgeborgen. De positie van het huidige centrale objectpunt wordt gecontroleerd in de functie *reposition*. Als die positie niet acceptabel is, dan wordt het centrum van het begrenzende prisma gebruikt als het nieuwe punt O; de nieuwe oogcoördinaten x_e, y_e, z_e worden dan voor alle hoekpunten berekend door de functie *viewing*. Terugkerend naar *reposition*, roepen we ook de functie *screencoor* aan, die zonodig de variabelen *xmin*, *xmax*, *ymin* en *ymax* (niet te verwarren met *xwmin* etc.) bijwerkt. Deze stellen de minimum en maximum waarden van de uitdrukkingen x_e/z_e en y_e/z_e voor, die in de perspectivische transformatie voorkomen; zij stellen ons in staat de belangrijke 'constanten' c_1, c_2 en d, die ook in de perspectivisch transformatie voorkomen, uit te rekenen. Met behulp van

$$Xrange = xmax - xmin$$
$$Yrange = ymax - ymin$$
$$Xcenter = (xmin + xmax)/2$$
$$Ycenter = (ymin + ymax)/2$$

en gebruik makend van zowel het centrum (*Xvp*_center, *Yvp*_center) als de afmetingen *Xvp*_range en *Yvp*_range van de gegeven viewport, hebben we

$$f_x = \frac{Xvp_\text{range}}{Xrange} \quad f_y = \frac{Yvp_\text{range}}{Yrange}$$

d = de kleinste van de twee waarden f_x en f_y
$c_1 = Xvpcenter - d \,.\, Xcenter$
$c_2 = Yvpcenter - d \,.\, Ycenter$

zoals meer in detail besproken wordt in mijn boek *PPCG*.

Uit het bovenstaande volgt dat we te maken hebben met zowel tweedimensionale als driedimensionale begrenzingen. Wat de laatstgenoemde betreft, we moeten

ervoor zorgen dat het begrenzende prisma (bepaald door *xwmin*, *xwmax*, *ywmin*, *ywmax*, *zwmin*, *zwmax*) wordt bijgewerkt wanneer een punt wordt opgegeven dat erbuiten ligt. De tweedimensionale begrenzingen worden vastgelegd door de variabelen *xmin*, *xmax*, *ymin*, *ymax*, die de viewport bepalen. Voor elk nieuw punt moeten we controleren dat het beeld ervan binnen deze tweedimensionale grenzen valt. In de functie *plotpoint* wordt de functie *screencoor* aangeroepen, die de globale variabele *inside* op nul zet als die tweedimensionale controle een negatieve uitslag heeft. In dat geval wordt functie *reposition* aangeroepen en er wordt eerst een driedimensionale controle uitgevoerd: als het huidige 'centrale objectpunt' O te ver van het exacte centrum van het begrenzende prisma af ligt wordt het bijgewerkt; voor alle hoekpunten wordt dan de functie *storepoint* aangeroepen om nieuwe oogcoördinaten x_e, y_e en z_e te berekenen. Ongeacht de uitkomst van de driedimensionale test berekent functie *reposition* de nieuwe waarden c_1, c_2 en d. Een hierop volgende aanroep van de functie *display* zal deze nieuwe waarden gebruiken om een geheel nieuw beeld op het scherm te brengen, waarbij alle beeldpunten binnen de viewport liggen. Het vernieuwen van het scherm kan dus al of niet gepaard gaan met het berekenen van nieuwe oogcoördinaten. Als deze berekening achterwege blijft verschijnt het nieuwe beeld veel vlugger dan wanneer hij wordt uitgevoerd. Toch zal het vernieuwen van het scherm, ook als alleen maar nieuwe schermcoördinaten berekend worden (en de oogcoördinaten ongewijzigd blijven) wat tijd kosten. Veronderstel nu dat we bezig zijn de cursor in een vaste richting te verplaatsen, waarbij we slechts kleine stapjes gebruiken, en dat we hiermee doorgaan nadat we de grens van de viewport hebben bereikt. Het zou dan vervelend zijn als bij elk van deze kleine stapjes het scherm helemaal vernieuwd zou worden. Daarom is hier wat op gevonden. Wanneer we het werkscherm (met de commando's op de rechter helft) gebruiken, wordt iedere nieuwe waarde van d gereduceerd met een 'tolerantiefactor', waarvoor ik het getal 0.85 heb gebruikt. Dit betekent dat, onmiddellijk na het vernieuwen van het scherm, de grootte van het beeld gelijk is aan slechts 0.85 maal de maximale grootte die de viewport toestaat. Zodoende kan de cursor opnieuw, in dezelfde richting als tevoren, enkele stappen maken voordat hij opnieuw de viewport bereikt. De waarde 0.85 van deze 'tolerantiefactor' is vrij willekeurig gekozen, en u kunt hem desgewenst (in de functie *reposition*) wijzigen. Hoe groter hij is (mits kleiner dan 1) hoe vaker het scherm als gevolg van cursorbewegingen vernieuwd zal worden, dus u kunt hem verkleinen als het vernieuwen van het scherm naar uw smaak te vaak plaatsvindt. Maar de prijs die hiervoor moet worden betaald is dat onmiddellijk na het vernieuwen van het scherm de ongebruikte marges langs de vier zijden van de viewport groter worden: het beeld wordt dus kleiner.

U kunt zich afvragen of het werkelijk nodig is de tweedimensionale controle uit te voeren voor alle individuele punten; als we er immers voor zorgen dat elk hoekpunt van het begrenzende prisma een beeldpunt heeft dat binnen de viewport ligt dan zullen alle punten binnen dat prisma zeker een beeld binnen de viewport opleveren. Deze methode is inderdaad veilig, maar hij is helaas *te* veilig. Het gebruik ervan zou

voor veel voorwerpen een beeld opleveren dat veel kleiner is dan wat mogelijk is. Neem bijvoorbeeld een bol, waarvan het begrenzende prisma een kubus is waarin de bol precies past. Bij de meeste keuzen van het oogpunt zal de kubus meer ruimte op het scherm innemen dan de bol, dus het is niet goed de perspectivische transformatie van de te tekenen bol te baseren op de afmetingen van de niet te tekenen kubus. Bovendien zou deze methode dikwijls een vernieuwing van het scherm tot gevolg hebben voor punten (of cursorposities) die, hoewel vallend buiten het huidige begrenzende prisma, nog steeds een tweedimensionaal beeld hebben dat binnen de viewport ligt. Bij onze methode daarentegen wordt de positie van een punt vergeleken met de viewport in plaats van het begrenzende prisma. We staan toe dat het begrenzende prisma een beeld heeft dat niet binnen de viewport past; we hebben dit prisma alleen maar nodig om, wanneer dat vereist is, een nieuw objectpunt O te kunnen berekenen.

Bovenstaande opmerkingen vormen allerminst een volledige uitleg van programma D3D, maar zij betreffen wel de meest wezenlijke elementen ervan. Enige andere essentiële ingrediënten zijn driedimensionale rotaties, laag-niveau grafische routines en verborgen-lijn-eliminatie; een bespreking hiervan is te vinden in respectievelijk de paragrafen 3.7, 4.3 en 4.4. De module D3D zelf is opgenomen in Appendix A. Hij bestaat uit een groot aantal functies van betrekkelijk geringe omvang, waardoor de programmatekst zelf wellicht hulp kan bieden als u precies wilt nagaan hoe D3D werkt.

4.3 LAAG-NIVEAU GRAFISCHE FUNCTIES

Vijf van de zes hoofdstukken van mijn boek *Computer Graphics for the IBM PC* (van nu af aan afgekort tot *CGIP*) gaan over de ontwikkeling van een pakket, GRPACK, van laag-niveau grafische C-functies. De aard van die software verschilt van de meeste programma's in het onderhavige boek in de zin dat GRPACK in nauw verband staat met specifieke hardware en met bijzondere faciliteiten van de gebruikte C-compiler. We maken dus onderscheid tussen machine-gerichte of laag-niveau software en applicatiegerichte software, die in ons geval meer wiskundig georiënteerd is en 'hoog-niveau' genoemd zou kunnen worden. (Bij gebruik van deze terminologie moeten de termen *laag* en *hoog* overigens niet worden geassocieerd met de begrippen 'gemakkelijk' en 'moeilijk'!) Toen ik *CGIP* schreef was ik me ervan bewust dat enkele laag-niveau grafische functies niet bijzonder elegant waren, wat het gevolg is van het feit dat de IBM PC zelf niet in alle opzichten elegant is. Maar we kunnen machine- en compilerafhankelijke aspecten niet altijd vermijden als we programma's uit de praktijk willen bespreken. De laag-niveau grafische functies zijn de bouwstenen voor hoog-niveau graphics; zowel het hoofdprogramma D3D als de verborgen-lijn module HLPFUN maakt gebruik van de functies die in deze paragraaf worden besproken.

We zullen hier een versie van GRPACK bespreken die iets gewijzigd is ten opzichte van die in *CGIP.* Er komen geen functies in voor die bedoeld zijn voor het vullen van een gesloten gebied en voor het tekenen van cirkels, omdat we deze faciliteiten in D3D niet nodig hebben. Belangrijker is dat de hier behandelde versie van GRPACK geaccepteerd wordt door de Turbo C compiler (terwijl de oorspronkelijke versie is gebaseerd op Lattice C). Een ander nieuw punt is dat GRPACK nu betere resultaten geeft op machines die werken met een Enhanced Graphics Adapter (EGA). De oorspronkelijke versie maakt onderscheid tussen de Hercules Graphics Adapter (HGA), resolutie 720 x 348, en de Color Graphics Adapter (CGA), resolutie 640 x 200, en kiest laatstgenoemde resolutie als EGA in gebruik is. In plaats hiervan kan GRPACK nu kiezen uit drie mogelijkheden, namelijk HGA, CGA (beide met de zojuist genoemde resoluties) en EGA, met resolutie 640 x 350.

Tenslotte is er nu ook de mogelijkheid om een 'hard copy' te verkrijgen op een Hewlett-Packard plotter. Dit gebeurt door gegevens te schrijven naar een file die gelezen kan worden door een ander programma, PLOTHP, dat in paragraaf 4.5 besproken zal worden. Daar dit een normale ASCII-file is, die de door de gebruiker opgegeven wereldcoördinaten bevat, kunnen we hem eventueel ook voor andere doeleinden gebruiken.

Naast GRPACK is er een alternatief, GRPACK1; deze module is gebaseerd op Turbo C versie 1.5, waarin grafische faciliteiten direct beschikbaar zijn; we zullen GRPACK1 later in deze paragraaf bespreken.

Omdat ik er niet van uit wil gaan dat u bekend bent met mijn boek *CGIP,* zullen we eerst bespreken hoe mijn laag-niveau grafische routines kunnen worden gebruikt. We maken onderscheid tussen reële schermcoördinaten in inches, aangeduid met kleine letters x en y, en geheeltallige pixelcoördinaten, aangeduid met de hoofdletters X en Y. De oorsprong van het reële coördinatenstelsel ligt linksonder op het scherm:

$$0 \leqq x \leqq xmax$$
$$0 \leqq y \leqq ymax$$

Tenzij *setprdim* is aangeroepen (zie hieronder), hebben we:

$$x_max = 10.0$$
$$y_max = 7.0$$

(Dus we nemen aan dat het beeldscherm 10 x 7 inch groot is; aangezien dit alleen maar een benadering is moet de term *inch*, met betrekking tot het beeldscherm, niet te letterlijk worden opgevat.)

De oorsprong ($X=0$, $Y=0$) van het pixelcoördinatenstelsel ligt in de linkerbovenhoek van het scherm en we hebben:

$$0 \leqq X \leqq X__max$$
$$0 \leqq Y \leqq Y__max,$$

waarin de maximum waarden van X en Y afhangen van het adaptertype dat gebruikt wordt:

	X__max	*Y__max*
CGA:	639	199
HGA:	719	347
EGA:	639	349

Net als de functies die we zullen gebruiken, worden de variabelen *x_max*, *y_max*, *X__max*, *Y__max*, en nog enkele andere die we hieronder zullen noemen, gedefinieerd in GRPACK.C en gedeclareerd in de headerfile GRPACK.H. Het moet sterk worden aanbevolen laatstgenoemde file te gebruiken; we schrijven daarom

```
#include "grpack.h"
```

in elk programma dat GRPACK (of GRPACK1) gebruikt.

Het volgende overzicht laat zien welke functies in GRPACK aanwezig zijn. (Zij zijn ook in GRPACK1 beschikbaar, maar sommige hebben daar, zoals we zullen zien, een iets afwijkend effect.) Argumenten *x* en *y*, geschreven in kleine letters, zijn van het type *float* en argumenten waarvan de naam met een hoofdletter *X* of *Y* begint zijn van het type *int.* Zoals wellicht bekend, kennen de grafische adapters van de IBM PC twee toestanden, meestal 'graphics mode' en 'text mode' genoemd. Wij zullen deze termen alleen qua spelling vernederlandsen en spreken over de *grafische mode* en de *tekstmode*.

initgr() Initialiseer en schakel over naar de grafische mode. Te gebruiken in de tekstmode.

endgr() Wacht tot er op een toets is gedrukt. Hierna wordt weer naar de tekstmode overgeschakeld. Een aanroep van deze functie (of van *to_text*) mag niet vergeten worden! Te gebruiken in de grafische mode.

to_text() Keer onmiddellijk terug naar de tekstmode. Te gebruiken in de grafische mode.

move(x, y) Beweeg een denkbeeldige pen naar punt (x, y) (waarbij x en y niet-negatieve reële coördinaten zijn, ten hoogste gelijk aan x_max, respectievelijk y_max. Te gebruiken in de grafische mode.

imove(X, Y) Beweeg de 'pen' naar punt (X, Y) (waarbij X en Y nietnegatieve integer pixelcoördinaten zijn, ten hoogste gelijk aan X_max respectievelijk Y_max). Te gebruiken in de grafische mode.

draw(x, y) Teken een recht lijnstuk van de huidige penpositie naar de nieuwe penpositie (x, y). Te gebruiken in de grafische mode.

idraw(X, Y) Teken een lijnstuk van oude penpositie naar penpositie (X, Y). Te gebruiken in de grafische mode.

draw_line(X1, Y1, X2, Y2) Teken een lijnstuk van $(X1, Y1)$ naar $(X2, Y2)$ (zonder de 'huidige penpositie' te veranderen). Te gebruiken in de grafische mode.

dot(X,Y) Plaats een stip in punt (X, Y). Te gebruiken in de grafische mode.

clearpage() Maak het grafische scherm schoon. (De term *page* heeft te maken met het feit dat we bij HGA de twee pagina's 0 en 1 kennen; hiervan gebruiken wij alleen pagina 1.) Te gebruiken in de grafische mode.

text(str) Geef de karakterstring *str* weer op het beeldscherm, te beginnen in de huidige penpositie (waarschijnlijk verkregen door een aanroep van *imove* of *move*). De 'huidige penpositie' wordt gelijk gemaakt aan de eerste positie op het scherm die op de string volgt. Te gebruiken in de grafische mode.

textXY(X, Y, str) Werkt net als *text(str)*, behalve ten aanzien van het beginpunt, dat nu (in pixelcoördinaten) expliciet wordt opgegeven. Ook wordt nu de 'huidige penpositie' niet gewijzigd. Te gebruiken in de grafische mode.

printgr(Xlo, Xhi, Ylo, Yhi) Druk de (grafische) inhoud van de 'viewport'

$$Xlo \leqq X \leqq Xhi, Ylo \leqq Y \leqq Yhi$$

af op een matrixprinter (op LPT1). Zie ook *setprdim()*. Te gebruiken in de grafische mode.

setprdim() 'Set print dimensions'. Als we deze functie vóór *initgr* aanroepen, dan worden aan *xmax* en *ymax* waarden toegekend die met de 'aspect ratio'

van de matrixprinter overeenkomen. Als gevolg hiervan zal de functie *printgr* uitvoer op de printer leveren met correcte afmetingen, zowel in de horizontale als in de verticale richting. Hierna zal bijvoorbeeld een cirkel (benaderd door een regelmatige veelhoek) worden afgedrukt als een cirkel en niet als een ellips. Helaas zal dan een cirkel op het scherm juist als een ellips verschijnen, dus *setprdim* moet niet worden aangeroepen als een optimaal resultaat op het beeldscherm vereist is. Te gebruiken in de tekstmode.

De volgende functies leveren een integer functiewaarde af:

IX(x) De integer pixelcoördinaat *X* die overeenkomt met de reële schermcoördinaat *x*. Te gebruiken in de grafische mode.

IY(y) De integer pixelcoördinaat *Y* die overeenkomt met de reële schermcoördinaat *y*. Te gebruiken in de grafische mode.

pixlit(X, Y) Geeft 1 terug als pixel (*X*, *Y*) verlicht is en 0 als hij donker is. Te gebruiken in de grafische mode.

iscolor() Geeft een code terug voor de grafische adapter die in gebruik is:

2 = EGA (nieuw)
1 = CGA
0 = HGA

Kan zowel in de tekstmode als in de grafische mode worden gebruikt.

Er zijn drie functies in GRPACK die eigenlijk niet echt als grafische functies beschouwd kunnen worden, hoewel ze te maken hebben met het beeldscherm. Zij kunnen alleen in de tekstmode worden gebruikt:

wrscr(row, column, str)
Schrijf de karakterstring *str* op het scherm, beginnend in de positie *row, column*. Het rijnummer *row* loopt van 0 t/m 24, het kolomnummer *column* van 0 tot 79. De positie in de linker bovenhoek van het scherm heeft positie (0, 0). Te gebruiken in de tekstmode.

settxtcursor(row, column)
Zet de tekstcursor (een knipperend streepje onderaan de regel) op positie *row, column*. (Zie ook *wrscr*.) Te gebruiken in de tekstmode.

clearscr() Maak het scherm schoon. Te gebruiken in de tekstmode.

(De drie laatstgenoemde functies *wrscr, settxtcursor* en *clearscr* komen niet voor in de oorspronkelijke versie van GRPACK, dus zij worden niet besproken in *CGIP.*)

We kunnen in de grafische mode iets van het scherm verwijderen door -1 toe te kennen aan de variabele *drawmode*, die gedeclareerd wordt in de header-file GRPACK.H en gewoonlijk de waarde 1 heeft. Als *drawmode* gelijk is aan -1, dan zullen de functies *draw, idraw, drawline* en *dot* de pixels donker maken. Er is een derde mogelijkheid: we kunnen *drawmode* gelijk aan 0 maken, wat maakt dat de pixels worden geïnverteerd; als zij donker zijn worden ze door de vier zojuist genoemde functies verlicht en als zij verlicht zijn worden ze donker gemaakt.

Zoals eerder vermeld is er een header-file, die als volgt gebruikt dient te worden:

```
#include "grpack.h"
```

Het gebruik hiervan moet worden aanbevolen in elk programma dat van GRPACK (of GRPACK1) gebruik maakt. De tekst van deze header-file volgt hieronder:

```
/* GRPACK.H: Header-file voor grafische functies.
*/

extern int in_textmode, X__max, Y__max, drawmode;
extern float x_max, y_max, horfact, vertfact;
extern FILE *fplot;
/* File voor uitgestelde grafische uitvoer. De file kan
   bijvoorbeeld door programma PLOTHP worden gelezen om
   uitvoer op het beeldscherm, op een matrixprinter, of
   op een HP-plotter te verkrijgen.
   De naam FILE wordt gedefinieerd in de header-file 'stdio.h',
   zodat deze, door middel van de regel #include <stdio.h>, aan
   de regel #include "grpack.h" moet voorafgaan.
*/
#ifndef __HUGE__
#error Gebruik 'huge' geh. model
#endif

void initgr(void);
void endgr(void);
void to_text(void);
void move(float x, float y);
void draw(float x, float y);
void imove(int X, int Y);
void idraw(int X, int Y);
void draw_line(int X1, int Y1, int X2, int Y2);
```

```
void dot(int X, int Y);
void clearpage(void);
void text(char *str);
void textXY(int X, int Y, char *str);
void printgr(int Xlo, int Xhi, int Ylo, int Yhi);
void setprdim(void);
int IX(float x);
int IY(float y);
int pixlit(int X, int Y);
int iscolor(void);
void wrscr(int line, int col, char *str);
void settxtcursor(int line, int col);
void clearscr(void);
```

Naast de genoemde variabelen *X__max*, *Y__max* en *drawmode* is er nog een nuttige globale variabele, eveneens van het type *int*:

in_textmode=1 als het systeem in de tekstmode is,
=0 als het systeem in de grafische mode is.

Als globale *float* variabelen hebben we naast *x_max* en *y_max* nog:

horfact =aantal pixels per inch in horizontale richting.
vertfact =aantal pixels per inch in verticale richting.

De waarden van *x_max*, *y_max*, *horfact* en *vertfact* zijn van toepassing op het beeldscherm, behalve als, vóór de aanroep van *initgr*(), de functie *setprdim*() wordt aangeroepen; in dat geval zijn zij van toepassing op de printer.

Als laatste globale variabelen moet de file pointer *fplot* te worden genoemd. Deze heeft normaal de (default-)waarde *NULL*, en wordt dan genegeerd. Als we er daarentegen een andere waarde aan hebben toegekend, verkregen met behulp van de standaardfunctie *fopen*, dan doet elke aanroep van de grafische functies *move* en *draw* iets extra's. Deze functies schrijven dan namelijk regels tekst in de file die via de genoemde file-pointer toegankelijk is. Elk van deze regels bevat drie getallen, namelijk de twee argumenten (*x* en *y*) van *move* en *draw* en een code 0 voor *move* of 1 voor *draw*, die aangeeft om welke van deze twee functies het gaat. Op deze manier verschijnt de lijntekening die we aan het maken zijn niet alleen op het beeldscherm maar hij komt ook beschikbaar in machine-leesbare vorm. In paragraaf 4.5 zullen we zien dat dit heel nuttig kan zijn, in het bijzonder als een Hewlett-Packard plotter beschikbaar is.

De tekst van GRPACK is hieronder weergegeven; de meeste functies erin worden

uitgelegd in mijn boek *CGIP.* Voor het geval u er elementen in mocht aantreffen die u, in verband met bijzondere hardware of software niet kunt gebruiken, is het misschien goed alvast te vermelden dat er een alternatief is, namelijk GRPACK1, die hierna besproken zal worden.

```
/* GRPACK: Een grafisch pakket.
      Turbo C versie, Memory-model: Huge.

      Geschikt voor:
           Enhanced Graphics Adapter (EGA, 640 x 350)
           Color Graphics Adapter (CGA, 640 x 200)
           Hercules Graphics Adapter (HGA, 720 x 348)

      Zie ook:
      L. Ammeraal (1987). Computer Graphics for the
          IBM PC, Chichester: John Wiley & Sons.
*/
#ifndef __HUGE__
#error Gebruik 'huge' geh.model
#endif

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <process.h>
#include <string.h>

union REGS regs;

int in_textmode=1, X__max, Y__max, drawmode=1;

FILE *fplot=NULL;

float x_max=10.0, y_max=7.0, horfact, vertfact;

static int old_vid_state, X1, Y1, offset, adaptype,
   Ystart[350]; /* >= Ymax + 1 */

static long int startaddress;

static char
   lastchar,
   gtable[12] =
```

```
      {53, 45, 46,  7, 91, 2, 87, 87, 2,  3,  0,  0},
   ttable[12] =
      {97, 80, 82, 15, 25, 6, 25, 25, 2, 13, 11, 12},
   zeros[128]; /* impliciet op 0 gezet */

void to_text(void);
void textXY(int X, int Y, char *str);


int IX(float x) { return (int)(x*horfact+0.5); }
int IY(float y) { return Y__max-(int)(y*vertfact+0.5); }


static void error(char *str)
/* Geef foutboodschap en stop met de programma-uitvoering. */
{ if (!in_textmode) to_text();
  printf("%s\n", str); exit(1);
}


void clearpage(void)  /* Maak het grafische scherm schoon. */
{ int i, n;
  unsigned segsrc, offsrc, segdest;
  n = (adaptype == 1 ? 128 :
       adaptype == 0 ? 256 : 219);
                              /* EGA: 219 x 128 = 28032 */
  segsrc = FP_SEG(zeros); offsrc = FP_OFF(zeros);
  segdest = (adaptype == 2 ? 0xA000 : 0xB800);
  for (i=0; i<n; i++)
  movedata(segsrc, offsrc, segdest, i << 7, 128);
}


int iscolor(void)
/* Zoek uit welke adapter gebruikt wordt:
    2 = EGA
    1 = CGA
    0 = HGA
   -1 = niet-grafische adapter
*/
{ char ch0, ch1, x;
  if (peekb(0x40, 0x87)) return 2;
  /* EGA, zie Dr. Dobb's Journal, November 1987, p.34 */
  int86(0x11, &regs, &regs);
```

```
  if ((regs.x.ax & Øx3Ø) != Øx3Ø) return 1; /* CGA */
  outport(Øx3BF, 3);
    /* Configuration switch                              */
  chØ = *(char *)ØxB8ØØØØØØ;
    /* Probeer chØ uit het schermgeheugen te lezen      */
  ch1 = chØ ^ ØxFF;
    /* Vind een waarde verschillend van chØ             */
  *(char *)ØxB8ØØØØØØ = ch1;
    /* Probeer dit in het schermgeheugen te schrijven  */
  x = *(char *)ØxB8ØØØØØØ;
    /* Probeer laatstgenoemde waarde te lezen           */
  *(char *)ØxB8ØØØØØØ = chØ;
    /* Herstel de oude waarde  chØ                      */
  return (x == ch1 ? Ø : -1);
    /* Is de geschreven waarde ook gelezen?             */
}

static void setgrcon(int adaptype)
/* Stel grafische constanten in. */
{ int Y, c1, c2, c3;
  startaddress = (adaptype == 2 ? ØxAØØØØØØØ : ØxB8ØØØØØØ);
  if (adaptype == 2)
  { X__max = 639; Y__max = 349;
    for (Y=Ø; Y<=Y__max; Y++) Ystart[Y] = 8Ø * Y;
    return;
  }
  if (adaptype == 1)
  { X__max = 639; Y__max = 199;
    c1 = 1; c2 = 8Ø; c3 = 1;
  } else
  { X__max = 719; Y__max = 347;
    c1 = 3; c2 = 9Ø; c3 = 2;
  }
  for (Y=Ø; Y<=Y__max; Y++)
  Ystart[Y] = Øx2ØØØ*(Y&c1) + c2*(Y>>c3);
}


static int grbrfun(void)
/* Wordt gebruikt door 'ctrlbrk' om op te geven wat er
   bij een 'console-break' moet gebeuren.
*/
{ to_text(); return Ø;
}
```

```
static void initcgaorega(int mode)
/* Schakel over naar de grafische mode (CGA of EGA).   */
{ regs.h.ah = 15; /* Vraag de huidige videotoestand op */
  int86(0x10, &regs, &regs);
  old_vid_state = regs.h.al;
  regs.h.ah = 0;          /* Naar grafische mode       */
  regs.h.al = mode;
  /* mode = 6:  CGA, 640 x 200, zwart/wit   */
  /* mode = 16: EGA, 640 x 350, zwart/wit   */
  int86(0x10, &regs, &regs);
}


static void initmongr(void)
/* Schakel over naar de grafische mode (HGA). */
{ static int firstcall=1;
  int i;
  outport(0x3B8, 0x82);
  for (i=0; i<12; i++)
  { outport(0x3B4, i);
    outport(0x3B5, gtable[i]);
  }
  if (firstcall) { firstcall=0; clearpage(); }
  outport(0x3B8, 0x8A);
}


void initgr(void)
/* Grafische initialisatie. */
{ if (!in_textmode)
    error("initgr wordt aangeroepen in de grafische mode");
  adaptype = iscolor();
  if (adaptype < 0) error("Verkeerde display adapter");
  ctrlbrk(grbrfun);    /* Zet break trap, Turbo C   */
  if (adaptype == 2) initcgaorega(16); /* EGA */ else
  if (adaptype == 1) initcgaorega(6);  /* CGA */ else
  /* adaptype == 0 */ initmongr();     /* HGA */
  setgrcon(adaptype);
  in_textmode=0;
  horfact = X__max/x_max; vertfact = Y__max/y_max;
}
static void endcolgr(void)
/* Keer terug naar tekstmode (CGA of EGA). */
{ regs.h.ah = 0; regs.h.al = old_vid_state;
  int86(0x10, &regs, &regs);
}
```

```
static void endmongr(void)
/* Keer terug naar tekstmode (HGA). */
{ int i;
  char *source;
  unsigned segsrc, offsrc;
  outport(0x3B8, 0);
  for (i=0; i<12; i++)
  { outport(0x3B4, i);
    outport(0x3B5, ttable[i]);
  }
  source =
  "\40\7\40\7\40\7\40\7\40\7\40\7\40\7\40\7";
  segsrc = FP_SEG(source); offsrc = FP_OFF(source);
  for (i=0; i<256; i++)
    movedata(segsrc, offsrc, 0xB000, i << 4, 16);
  outport(0x3B8, 0x08);
}


static int txtbrfun(void)
/* Zie to_text. */
{ return 0;
}


void to_text(void)
/* Keer onmiddellijk terug naar tekstmode. */
{ if (in_textmode)
    error("endgr of to_text wordt aangeroepen in tekstmode");
  if (adaptype) endcolgr(); /* CGA of EGA */
           else endmongr(); /* HGA        */
  in_textmode = 1;
  ctrlbrk(txtbrfun);
  /* Herstel default break interrupt handler */
}


void endgr(void)
/* Wacht tot er op een toets is gedrukt en keer
   terug naar tekstmode.
*/
{ getch();
  to_text();
}
```

```
void dot(int X, int Y)
/* Maak een pixel licht of donker. */
{ int pattern;
  offset = Ystart[Y] + (X>>3);
  lastchar = *(char *)(startaddress + offset);
                                                /* Turbo C  */
  pattern = Øx8Ø >> (X&7);
  if (drawmode == 1) lastchar |= pattern; else
  if (drawmode == -1) lastchar &= (~pattern);
                  else lastchar ^=pattern;
  *(char *)(startaddress + offset) = lastchar;
}


static void checkbreak(void)
/* Houd Ctrl-Break in de gaten. */
{ char ch;
  if (kbhit()) { ch = getch(); kbhit(); ungetch(ch); }
}


void draw_line(int X1, int Y1, int X2, int Y2)
/* Teken het lijnstuk van (X1, Y1) naar (X2, Y2);
   X1, Y1, X2, Y2 zijn pixelcoordinaten.
*/
{ int X, Y, T, E, dX, dY, denom, Xinc = 1, Yinc = 1,
  vertlonger = Ø, aux;
  checkbreak(); /* Om DOS te laten testen op console break */
  if (in_textmode)
    error("Niet in grafische mode (roep initgr aan)");
  dX = X2 - X1; dY = Y2 - Y1;
  if (dX < Ø) {Xinc = -1; dX = -dX;}
  if (dY < Ø) {Yinc = -1; dY = -dY;}
  if (dY > dX)
  { vertlonger = 1; aux = dX; dX = dY; dY = aux;
  }
  denom = dX << 1;
  T = dY << 1;
  E = -dX; X = X1;
  Y = Y1;
  if (vertlonger)
  while (dX-- >= Ø)
 { dot(X, Y);
  if ((E += T) > Ø)
  { X += Xinc; E -= denom;
  }
```

```
      Y += Yinc;
    } else
    while (dX-- >= 0)
    { dot(X, Y);
      if ((E += T) > 0)
      { Y += Yinc; E -= denom;
      }
      X += Xinc;
    }
    if (drawmode == 0) dot(X1, Y1);
}

static void fatal(void)
/* Trek een diagonaal en wacht dan tot er op een toets is
   gedrukt; ga daarna terug naar tekstmode.
*/
{ draw_line(0, Y__max,  X__max, 0); endgr();
}


static void check(int X, int Y)
/* Als punt (X, Y) buiten de grenzen van het scherm ligt,
   roep dan 'fatal' aan, druk de incorrecte coordinaten af
   en stop.
*/
{ if (X < 0 || X > X__max || Y < 0 || Y > Y__max)
  { fatal();
    printf(
"Punt buiten scherm (X en Y zijn pixelcoordinaten):\n");
    printf("X = %d       Y = %d\n", X, Y);
    printf("x = %10.3f   y = %10.3f\n",
    X/horfact, (Y__max-Y)/vertfact);
    printf(
    "X__max = %d  Y__max = %d  x_max = %f y_max = %f\n",
     X__max, Y__max, x_max, y_max);
    printf("horfact = %f  vertfact = % f\n",
            horfact, vertfact);
    exit(1);
  }
}


void move(float x, float y)
/* Verplaats het huidige punt naar (x, y);
   x en y zijn schermcoordinaten.
*/
```

```
{ int XX, YY;
  XX = IX(x); YY = IY(y); check(XX, YY);
  if (fplot != NULL && (XX != X1 || YY != Y1))
    fprintf(fplot, "%6.3f %6.3f 0\n", x, y);
  X1 = XX; Y1 = YY;
}


void draw(float x, float y)
/* Teken een lijnstuk vanaf het huidige punt naar (x, y). */
{ int X2, Y2;
  X2 = IX(x); Y2 = IY(y); check(X2, Y2);
  draw_line(X1, Y1, X2, Y2);
  if (fplot != NULL)
    fprintf(fplot, "%6.3f %6.3f 1\n", x, y);
  X1 = X2; Y1 = Y2;
}


int pixlit(int X, int Y)
/* Informeer of pixel (X, Y) verlicht is. */
{ int pattern;
  offset= Ystart[Y] + (X>>3);
  pattern = 0x80 >> (X&7);
  lastchar = *(char *)(startaddress + offset);
  return ((lastchar & pattern) != 0);
}

static void prchar(char ch)
/* Zend ch naar de parallelle printerpoort LPT1.          */
{ regs.x.dx=0;          /* Printerselectie /              */
  regs.h.ah=0;          /* Zend byte van AL naar printer  */
  regs.h.al=ch;         /* Byte bestemd voor printer      */
  int86(0x17, &regs, &regs);
}

void printgr(int Xlo, int Xhi, int Ylo, int Yhi)
/* Print inhoud van rechthoek op matrixprinter.      */
{ int n1, n2, ncols, i, X, Y, val, Xhi1;
  char line[720];                      /* 720 >= X__max */
  regs.x.dx = 0;                       /* LPT1          */
  regs.h.ah = 2;  /* Service 2: Get Printer Status    */
  int86(0x17, &regs, &regs);  /* Printer services     */
  if ((regs.h.ah & 8) == 8)   /* Printer status in AH */
  { if (in_textmode) printf("Tekstmode (printgr)"); else
    textXY(0, Y__max - 15, "Printer niet gereed");
```

```
    return;
  }
  prchar(27); prchar('1'); /* Line spacing 7/72 inch */
  for (i=Ylo; i<=Yhi; i+=7)
  { checkbreak();
     /* Laat DOS testen op console break  */
    Xhi1 = Xlo-1;
    for (X=Xlo; X<=Xhi; X++)
    { val=0;
      for (Y=i; Y<i+7; Y++)
      { val <<= 1; val |= (Y>Yhi ? 0 : pixlit(X, Y));
      }
      line[X] = val;
      if (val) Xhi1 = X;
    }
    ncols = Xhi1-Xlo+1;
    if (ncols)
    { n1=ncols%256; n2=ncols/256;
      prchar(27); prchar('L'); prchar(n1); prchar(n2);
      for (X=Xlo; X<=Xhi1; X++) prchar(line[X]);
    }
    prchar('\n');
  }
  prchar(27); prchar('@');
}

void setprdim(void)
/* Deze functie geeft zodanige waarden aan x_max en y_max
   dat de maten op de printer goed uitkomen.
*/
{ extern float x_max, y_max;
  extern int X__max, Y__max;
  setgrcon(iscolor());
  x_max = (X__max + 1)/120.0; y_max=(Y__max + 1)/72.0;
}


#define NASCII 128
static char chlist[NASCII][11]=
{
 {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0},
 {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0},
 {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0}, {0},
/* */ {0},
/*!*/ {0x10, 0x38, 0x38, 0x38, 0x10, 0x10, 0x10, 0, 0x10},
```

```
/*"*/ {0x48, 0x48, 0x48},
/*#*/ {0x24, 0x24, 0x64, 0xFE, 0x44, 0xFE, 0x4C, 0x48,
       0x48},
/*$*/ {0x10, 0x7C, 0xD0, 0xD0, 0x7C, 0x16, 0x16, 0x7C,
       0x10},
/*%*/ {0xC2, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
       0x86},
/*&*/ {0x30, 0x48, 0x48, 0x30, 0x50, 0x92, 0x8A, 0x8C,
       0x72},
/*'*/ {0x18, 0x18, 0x18, 0x10},
/*(*/ {0x08, 0x10, 0x20, 0x20, 0x20, 0x20, 0x20, 0x10,
       0x08},
/*)*/ {0x40, 0x20, 0x10, 0x10, 0x10, 0x10, 0x10, 0x20,
       0x40},
/***/ {0, 0x82, 0x44, 0x28, 0xFE, 0x28, 0x44, 0x82},
/*+*/ {0, 0x10, 0x10, 0x10, 0xFE, 0x10, 0x10, 0x10},
/*,*/ {0, 0, 0, 0, 0, 0, 0, 0x30, 0x30, 0x10, 0x20},
/*-*/ {0, 0, 0, 0, 0xFE},
/*.*/ {0, 0, 0, 0, 0, 0, 0, 0x30, 0x30},
/*/*/ {0, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80},
/*0*/ {0x38, 0x6C, 0x44, 0x44, 0x44, 0x44, 0x44, 0x6C,
       0x38},
/*1*/ {0x10, 0x30, 0x50, 0x10, 0x10, 0x10, 0x10, 0x10,
       0x38},
/*2*/ {0x7C, 0xC6, 0x02, 0x06, 0x0C, 0x18, 0x30, 0x60,
       0xFE},
/*3*/ {0x7C, 0xC6, 0x02, 0x06, 0x0C, 0x06, 0x02, 0xC6,
       0x7C},
/*4*/ {0x04, 0x0C, 0x1C, 0x34, 0x64, 0xC4, 0xFE, 0x04,
       0x04},
/*5*/ {0xFE, 0x80, 0x80, 0xFC, 0x06, 0x02, 0x02, 0xC6,
       0x7C},
/*6*/ {0x7C, 0xC6, 0x80, 0xFC, 0xC6, 0x82, 0x82, 0xC6,
       0x7C},
/*7*/ {0xFE, 0x02, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0,
       0x80},
/*8*/ {0x7C, 0xC6, 0x82, 0xC6, 0x7C, 0xC6, 0x82, 0xC6,
       0x7C},
/*9*/ {0x7C, 0xC6, 0x82, 0xC2, 0x7E, 0x02, 0x02, 0x06,
       0x7C},
/*:*/ {0, 0x30, 0x30, 0, 0, 0, 0, 0x30, 0x30},
/*;*/ {0, 0, 0x30, 0x30, 0, 0, 0, 0x30, 0x30, 0x10, 0x20},
/*<*/ {0x04, 0x08, 0x10, 0x20, 0x40, 0x20, 0x10, 0x08,
       0x04},
/*=*/ {0, 0, 0, 0, 0xFC, 0, 0, 0xFC},
```

```
/*>*/ {0x40, 0x20, 0x10, 0x08, 0x04, 0x08, 0x10, 0x20,
       0x40},
/*?*/ {0x78, 0xCC, 0x84, 0x0C, 0x18, 0x10, 0x10, 0, 0x10},
/*@*/ {0x7C, 0xC6, 0x8E, 0x92, 0x92, 0x92, 0x8C, 0xC0,
       0x7C},
/*A*/ {0x10, 0x38, 0x6C, 0xC6, 0x82, 0x82, 0xFE, 0x82,
       0x82},
/*B*/ {0xFC, 0x86, 0x82, 0x86, 0xFC, 0x86, 0x82, 0x86,
       0xFC},
/*C*/ {0x7C, 0xC6, 0x80, 0x80, 0x80, 0x80, 0x80, 0xC6,
       0x7C},
/*D*/ {0xFC, 0x86, 0x82, 0x82, 0x82, 0x82, 0x82, 0x86,
       0xFC},
/*E*/ {0xFE, 0x80, 0x80, 0x80, 0xF8, 0x80, 0x80, 0x80,
       0xFE},
/*F*/ {0xFE, 0x80, 0x80, 0x80, 0xFC, 0x80, 0x80, 0x80,
       0x80},
/*G*/ {0x7C, 0xC6, 0x82, 0x80, 0x80, 0x8E, 0x82, 0xC6,
       0x7C},
/*H*/ {0x82, 0x82, 0x82, 0x82, 0xFE, 0x82, 0x82, 0x82,
       0x82},
/*I*/ {0x38, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
       0x38},
/*J*/ {0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0xC6,
       0x7C},
/*K*/ {0x86, 0x8C, 0x98, 0xB0, 0xE0, 0xB0, 0x98, 0x8C,
       0x86},
/*L*/ {0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
       0xFE},
/*M*/ {0x82, 0xC6, 0xEE, 0xBA, 0x92, 0x82, 0x82, 0x82,
       0x82},
/*N*/ {0x82, 0xC2, 0xE2, 0xA2, 0xB2, 0x9A, 0x8E, 0x86,
       0x82},
/*O*/ {0x7C, 0xC6, 0x82, 0x82, 0x82, 0x82, 0x82, 0xC6,
       0x7C},
/*P*/ {0xFC, 0x86, 0x82, 0x86, 0xFC, 0x80, 0x80, 0x80,
       0x80},
/*Q*/ {0x7C, 0xC6, 0x82, 0x82, 0x82, 0x82, 0x92, 0xD6, 0x7C,
       0x08, 0x08},
/*R*/ {0xFC, 0x86, 0x82, 0x86, 0xFC, 0x90, 0x98, 0x8C,
       0x86},
/*S*/ {0x7C, 0xC6, 0x80, 0xC0, 0x7C, 0x06, 0x02, 0xC6,
       0x7C},
/*T*/ {0xFE, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
       0x10},
```

```
/*U*/ {0x82, 0x82, 0x82, 0x82, 0x82, 0x82, 0x82, 0xC6,
       0x7C},
/*V*/ {0x82, 0x82, 0x82, 0xC6, 0x44, 0x6C, 0x28, 0x38,
       0x10},
/*W*/ {0x82, 0x82, 0x82, 0x92, 0x92, 0xBA, 0xEE, 0x6C,
       0x44},
/*X*/ {0xC6, 0x44, 0x6C, 0x28, 0x38, 0x28, 0x6C, 0x44,
       0xC6},
/*Y*/ {0x82, 0x82, 0xC6, 0x6C, 0x38, 0x10, 0x10, 0x10,
       0x10},
/*Z*/ {0xFE, 0x04, 0x0C, 0x18, 0x10, 0x30, 0x60, 0x40,
       0xFE},
/*[*/ {0x7C, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40,
       0x7C},
/*\*/ {0, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02},
/*]*/ {0x78, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
       0x78},
/*^*/ {0x10, 0x28, 0x44, 0x82},
/*_*/ {0, 0, 0, 0, 0, 0, 0, 0, 0xFE},
/*`*/ {0x40, 0x20, 0x10, 0x08},
/*a*/ {0, 0, 0, 0x7C, 0x06, 0x7E, 0xC2, 0xC2, 0x7E},
/*b*/ {0x80, 0x80, 0x80, 0xFC, 0x86, 0x82, 0x82, 0x86,
       0xFC},
/*c*/ {0, 0, 0, 0x7C, 0xC6, 0x80, 0x80, 0xC6, 0x7C},
/*d*/ {0x04, 0x04, 0x04, 0x7C, 0xC4, 0x84, 0x84, 0xC4,
       0x7C},
/*e*/ {0, 0, 0, 0x7C, 0xC6, 0xFE, 0x80, 0xC0, 0x7C},
/*f*/ {0x1C, 0x30, 0x20, 0xFC, 0x20, 0x20, 0x20, 0x20,
       0x20},
/*g*/ {0, 0, 0, 0x7A, 0xCE, 0x82, 0x82, 0xC2, 0x7E, 0x06,
       0x7C},
/*h*/ {0x80, 0x80, 0x80, 0xFC, 0xC6, 0x82, 0x82, 0x82,
       0x82},
/*i*/ {0, 0x30, 0, 0x30, 0x10, 0x10, 0x10, 0x10, 0x38},
/*j*/ {0, 0x0C, 0, 0x0C, 0x04, 0x04, 0x04, 0x04, 0x04, 0xCC,
       0x78},
/*k*/ {0x40, 0x40, 0x40, 0x46, 0x4C, 0x78, 0x58, 0x4C,
       0x46},
/*l*/ {0x30, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
       0x38},
/*m*/ {0, 0, 0, 0xEC, 0x92, 0x92, 0x92, 0x92, 0x92},
/*n*/ {0, 0, 0, 0xBC, 0xC6, 0x82, 0x82, 0x82, 0x82},
/*o*/ {0, 0, 0, 0x7C, 0xC6, 0x82, 0x82, 0xC6, 0x7C},
/*p*/ {0, 0, 0, 0xFC, 0x86, 0x82, 0x82, 0x86, 0xFC, 0x80,
       0x80},
```

```
/*q*/ {0, 0, 0, 0x7C, 0xC4, 0x84, 0x84, 0xC4, 0x7C, 0x04,
       0x06},
/*r*/ {0, 0, 0, 0xBC, 0xE6, 0x80, 0x80, 0x80, 0x80},
/*s*/ {0, 0, 0, 0x7C, 0xC0, 0x7C, 0x06, 0x06, 0x7C},
/*t*/ {0, 0x40, 0x40, 0xF0, 0x40, 0x40, 0x40, 0x66, 0x3C},
/*u*/ {0, 0, 0, 0x84, 0x84, 0x84, 0x84, 0xC4, 0x7E},
/*v*/ {0, 0, 0, 0x82, 0x82, 0xC6, 0x6C, 0x38, 0x10},
/*w*/ {0, 0, 0, 0x82, 0x82, 0x92, 0xBA, 0xEE, 0x44},
/*x*/ {0, 0, 0, 0xC6, 0x6C, 0x28, 0x38, 0x6C, 0xC6},
/*y*/ {0, 0, 0, 0x82, 0x82, 0x82, 0x82, 0xC6, 0x7E, 0x06,
       0x7C},
/*z*/ {0, 0, 0, 0xFE, 0x0C, 0x18, 0x30, 0x60, 0xFE},
/*{*/ {0x10, 0x20, 0x20, 0x20, 0x40, 0x20, 0x20, 0x20,
       0x10},
/*|*/ {0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
       0x10},
/*}*/ {0x20, 0x10, 0x10, 0x10, 0x08, 0x10, 0x10, 0x10,
       0x20},
/*~*/ {0, 0, 0, 0x60, 0x92, 0x0C},
 {0}};


void textXY(int X, int Y, char *str)
/* Geef in grafische mode string str op het scherm
   weer, beginnend op positie (X, Y).
*/
{ char *p;
  int offset, i, j, len, hpos, vpos;
  len=strlen(str);
  hpos = X>>3;
  check(8*(hpos+len)-1, Y+10);
  for (i=0; i<len; i++)
  { p=chlist[str[i]];
    for (j=0; j<11; j++)
    { vpos=Y+j;
      offset= Ystart[vpos] + hpos + i;
      *(char *)(startaddress + offset) = p[j];
    }
  }
}

void text(char *str)
/* Geeft string  str  weer op het scherm, beginnend op
   de huidige positie (X1, Y1).
```

```
*/
{ textXY(X1, Y1, str);
  X1 += strlen(str) * 8;
  check(X1, Y1);
}

void imove(int X, int Y)
/* Laat (X, Y) het nieuwe huidige punt zijn. */
{ check(X, Y); X1=X; Y1=Y;
}

void idraw(int X, int Y)
/* Teken een lijnstuk van het huidige punt naar (X, Y)  */
{ check(X, Y);
  draw_line(X1, Y1, X, Y);
  X1=X; Y1=Y;
}

#define B000 (char *)0xB0000000
#define B800 (char *)0xB8000000

void wrscr(int row, int col, char *str)  /* In tekstmode */
/*  Schrijf, in tekstmode, karakters in het schermgeheugen.
    (row = 0, 1, ..., 24;  col = 0, 1, ..., 79)
*/
{ static char *displayptr;
  static int first=1;
  int i, offset;
  char *p;
  if (first)
  { displayptr = (iscolor() ? B800 : B000);
    first=0;
  }
  offset = row*160+col*2;
  p = displayptr+offset;
  for (i=0; str[i]; i++) p[i << 1] = str[i];
}

void clearscr(void)
/* Te gebruiken in tekstmode.
   Verwar deze functie niet met clearpage, die in
   grafische mode wordt gebruikt.
*/
```

```
{ static unsigned segdst;
  static first=1;
  int i;
  char *source;
  unsigned segsrc, offsrc;
  if (first)
  { segdst = (iscolor() ? 0xB800 : 0xB000);
    first = 0;
  }
  source =
  "\40\7\40\7\40\7\40\7\40\7\40\7\40\7\40\7";
  segsrc = FP_SEG(source); offsrc = FP_OFF(source);
  for (i=0; i<256; i++)
    movedata(segsrc, offsrc, segdst, i << 4, 16);
}

void settxtcursor(int row, int col)
/* Zet, in tekstmode, de cursor op de gewenste plaats. */
{ regs.h.dh = row;
  regs.h.dl = col;
  regs.h.ah = 2;
  regs.h.bh = 0;
  int86(0x10, &regs, &regs);
}
```

Zoals u waarschijnlijk bekend zal zijn, betekent het woord *static*, geschreven aan het begin van een functiedefinitie, dat de desbetreffende functie lokaal is in de module waarin hij is gedefinieerd. Zulke functies zijn alleen voor intern gebruik bestemd en niet beschikbaar voor de gebruiker. Alle functies in GRPACK die we niet hebben besproken zijn *static* gemaakt; op deze manier kunnen we ze gemakkelijk onderscheiden van de andere functies, die in gebruikersprogramma's kunnen worden aangeroepen. Als we (onopzettelijk) dezelfde namen voor functies van onszelf gebruiken dan is geen verwarring mogelijk, omdat de *static* functies niet aan de linker worden bekendgemaakt.

Juist toen ik de eerste opzet voor dit boek klaar had, ontving ik van Borland Internationational (vermoedelijk als een erg royale reactie op het melden van twee foutjes die ik in de eerste compilerversie ontdekt had) versie 1.5 van Turbo C. Deze nieuwe versie biedt schitterende grafische faciliteiten, die in plaats van GRPACK gebruikt zouden kunnen worden. Hoewel GRPACK bevredigend werkt op de machines die ik ken, realiseer ik me dat ik bij het testen van mijn grafische routines op huidige en toekomstige machinetypen niet kan concurreren met Borland;

evenmin kan ik mijn werk over de wereld verspreiden op de manier zoals Borland dat doet, dus ik heb de mogelijkheid overwogen om mijn GRPACK routines af te schaffen en radicaal over te stappen op de nieuwe functies van Turbo C. Gebruikers van oudere Turbo C versies zullen vroeg of laat op een nieuwere versie overstappen, dus de eis dat versie 1.5 gebruikt moet worden zou niet zo'n groot bezwaar zijn. Het wilde mij evenwel voorkomen dat er enkele aspecten van Turbo C zijn die misschien niet door alle gebruikers op prijs worden gesteld. In verband met de algemeenheid, gebruiken de Turbo C routines aanzienlijk meer geheugenruimte en sommige van die routines zijn langzamer dan die van GRPACK. Laatstgenoemd pakket is heel klein en als een klein pakket alles waaraan we behoefte hebben bevredigend doet dan zouden we het wel eens kunnen prefereren boven een meer omvangrijk pakket. Bij GRPACK wordt alles uitgedrukt in heel lage, dat wil zeggen machinegerichte, functies, zoals *int86* (voor software-interrupts) en *outport* (voor het adresseren van I/Opoorten), wat bekende faciliteiten zijn voor iedereen de goed bekend is met het programmeren van de IBM PC. Als gevolg hiervan zijn deze faciliteiten ook beschikbaar voor gebruikers van andere C-compilers. Dit laatste is belangrijk, want er bestaan heel wat goede C-compilers en het is niet waarschijnlijk dat alle IBM-PC-gebruikers die in C programmeren zullen overstappen op Turbo C. In het kort gezegd, de grafische functies van GRPACK zijn transparant, terwijl die van Turbo C gebruikt moeten worden als een 'black box'.

Misschien is wel de belangrijkste reden om GRPACK niet meteen maar af te schaffen dat het gebruik ervan leidt to uitvoerbare programma's (xxx.EXE-files) die lopen op machines met verschillende grafische adapters, zonder dat een 'installatieprocedure' of een stel 'driver files' vereist is. In tegenstelling hiermee, werkt de nieuwe grafische functie *initgraph* (die in Turbo C versie 1.5 wordt gebruikt om van tekstmode naar grafische mode over te schakelen) normaal zo dat de driver voor de geïnstalleerde grafische adapter pas tijdens de uitvoering wordt geladen. Dit betekent dat een algemeen bruikbaar uitvoerbaar programma, dat, ongeacht de grafische adapter, op elke machine moet werken, vergezeld moet gaan van verschillende driver-files. In plaats hiervan kunnen we ook een utility-programma (*BGIOBJ*) gebruiken om zulke driver-files naar objectfiles te converteren, waarna deze door de linker met onze programma's kunnen worden samengevoegd; maar in dat geval moeten we alle drivers erbij betrekken die eventueel nodig kunnen zijn. Omdat dit enigszins ingewikkeld is en in verband met de eerder genoemde andere punten lijkt het me niet zo slecht om ons aan GRPACK houden zolang dit aan onze behoeften voldoet.

Zoals opgemerkt aan het begin van deze paragraaf, zijn de laag-niveau grafische functies bijzonder essentieel: al onze grafische toepassingen op hoger niveau maken er immers gebruik van. Het is daarom geen verspilling van tijd als we een tweede oplossing aandragen voor het cruciale probleem van het tekenen van lijnstukken en het weergeven van karakters op het beeldscherm. Een keten is zo sterk als zijn zwakste schakel en als deze schakel GRPACK zou zijn (wat in verband met de

apparatuurafhankelijke aspecten niet onwaarschijnlijk is), dan zou het mooi zijn als u deze zou kunnen vervangen. Bovendien zullen wellicht sommige lezers van dit boek zo enthousiast over de nieuwe grafische functies van Turbo C versie 1.5 zijn dat ze D3D alleen maar in combinatie met die functies willen gebruiken. Ik heb hiervóór de term 'schitterend' gebruikt en ik denk dat die functies dat ook zijn, ondanks enkele nadelen ten opzichte van GRPACK. Om slechts enkele mogelijkheden te noemen, er zijn verschillende text-fonts en -stijlen beschikbaar, we kunnen manipuleren met grafische 'viewports' en met 'windows' voor tekst, we kunnen kleuren gebruiken en gesloten gebieden vullen met verschillende patronen. Het leek mij daarom onverstandig, de nieuwe grafische faciliteiten van Turbo C in dit boek geheel ongebruikt te laten.

Omdat de modules D3D en HLPFUN nogal groot en complex zijn, is het niet aantrekkelijk hiervan twee versies in dit boek op te nemen, één voor GRPACK en één voor de nieuwe functies van Turbo C. In plaats hiervan kunnen we de functies van GRPACK uitdrukken in termen van de nieuwe functies. Op deze manier verkrijgen we GRPACK1, een pakket dat voor de gebruiker bijna identiek is met GRPACK, maar dat gebruik maakt van de nieuwe functies van Turbo C. Als we het compileren levert het de objectmodule GRPACK1.OBJ op, die GRPACK.OBJ kan vervangen. De andere modules (D3D, HLPFUN, TRAFO) kunnen nu ongewijzigd blijven, evenals de header-file GRPACK.H. Deze alternatieve versie, GRPACK1, gebaseerd op Turbo C versie 1.5, is hieronder opgenomen.

```
/* GRPACK1: Een grafisch pakket dat gebruik maakt van
            nieuwe functies, beschikbaar in
            Turbo C, versie 1.5.
            Memory model: Huge.
            Geschikt voor:
              Enhanced Graphics Adapter (EGA, 640 x 350)
              Color Graphics Adapter (CGA, 640 x 200)
              Hercules Graphics Adapter (HGA, 720 x 348)
            Zie ook GRPACK, besproken in
            L. Ammeraal: Computer Graphics for the IBM PC.
*/
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <process.h>
#include <string.h>
#include <alloc.h>
#include <graphics.h>

union REGS regs;
```

```
   int in_textmode=1, X__max, Y__max, drawmode=1;

   FILE *fplot=NULL;

   float x_max=10.0, y_max=7.0, horfact, vertfact;

   static int X1, Y1, foregroundcol,
      backgroundcol, g_driver, g_mode;


   static long int startaddress;

   void to_text(void);
   void textXY(int X, int Y, char *str);


int IX(float x) { return (int)(x*horfact+0.5); }
int IY(float y) { return Y__max-(int)(y*vertfact+0.5); }


static void error(char *str)
/* Geef foutboodschap weer en stop */
{ if (!in_textmode) to_text();
  printf("%s\n", str); exit(1);
}

void clearpage(void)
/* Maak het grafische scherm schoon. */
{ clearviewport();
}


int iscolor(void)
/* Zoek uit welke adapter wordt gebruikt. */
{ detectgraph(&g_driver, &g_mode);
  if (g_mode == MCGAHI) g_mode = MCGAMED;
  if (g_mode == ATT400HI) g_mode = ATT400MED;
  if (g_mode == VGAHI) g_mode = VGAMED;
  return g_driver == CGA || g_driver == MCGA ||
         g_driver == ATT400 ?
           1 /* 640 x 200 */ :
         g_driver == EGA || g_driver == EGA64 ||
         g_driver == EGAMONO || g_driver == VGA ?
           2 /* 640 x 350 */ :
         g_driver == HERCMONO ? 0 /* 720 x 348 */ : -1;
}
```

```
static void setgrcon(int adaptype)
/* Stel grafische constanten in. */
{ startaddress = (adaptype == 2 ? 0xA0000000 : 0xB8000000);
  if (adaptype == 2)
  { X__max = 639; Y__max = 349;   /* EGA */
  } else
  if (adaptype == 1)
  { X__max = 639; Y__max = 199;   /* CGA */
  } else
  { X__max = 719; Y__max = 347;   /* HGA */
  }
}

static int grbrfun(void)
/* Wordt gebruikt door 'ctrlbrk' om op te geven wat er
   bij een 'console-break' moet gebeuren.
*/
{ to_text(); return 0;
  /* Keer terug naar tekstmode.
  */
}


void initgr(void)  /* Grafische initialisatie */
{ static int again=0;
  int errcode;
  if (again) setgraphmode(g_mode); else
  { adaptype = iscolor(); /* Bepaal driver en mode */
    if (adaptype < 0) error("Verkeerde display-adapter");
    ctrlbrk(grbrfun);   /* Set break trap, Turbo C   */
    initgraph(&g_driver, &g_mode, "\\turboc");
    errcode = graphresult();
    if (errcode < 0)
    { printf("Turbo C 1.5 grafische error code: %d",
             errcode); exit(1);
    }
    again = 1;
    setgrcon(adaptype);
  }
  horfact = X__max/x_max; vertfact = Y__max/y_max;
  if (adaptype == 0) /* HGA */
  { setvisualpage(1); setactivepage(1);
  }
  in_textmode=0;
  foregroundcol=getcolor();
```

```
  backgroundcol=getbkcolor();
}

static int txtbrfun(void)
{ return 0;
}
void to_text(void)
/* Keer terug naar tekstmode. */
{ if (!in_textmode) restorecrtmode();
  in_textmode = 1;
  ctrlbrk(txtbrfun);
  /* Herstel default break interrupt handler. */
}


void endgr(void)
/* Wacht tot er op een toets is gedrukt en keer
   terug naar tekstmode.
*/
{ getch();
  to_text();
}


void dot(int X, int Y)
/* Maak een pixel licht of donker. */
{ putpixel
  (X, Y,
    (drawmode == 0 ?
      ( getpixel(X, Y) == foregroundcol ?
        backgroundcol : foregroundcol
      ) :
      ( drawmode == 1 ? foregroundcol : backgroundcol
      )
      )    /* drawmode =  1: teken positief */
    );     /* drawmode = -1: teken negatief */
  }        /* drawmode =  0: inverteer      */


  void checkbreak(void)
  { char ch;
    if (kbhit()) { ch = getch(); kbhit(); ungetch(ch); }
  }

  void draw_line(int X1, int Y1, int X2, int Y2)
  /* Teken een lijnstuk van (X1, Y1) naar (X2, Y2);
```

```
   X1, Y1, X2, Y2 zijn pixelcoordinaten.
*/
{ if (drawmode == 1) line(X1, Y1, X2, Y2);
  /* 'drawmode'-waarden Ø en -1 niet geimplementeerd voor
      deze functie. Zie ook dot().
  */
}


static void fatal(void)
/* Trek een diagonaal, wacht dan tot er op een toets is
   gedrukt en keer daarna terug naar tekstmode.
*/
{ draw_line(Ø, Y__max,  X__max, Ø); endgr();
}

static void check(int X, int Y)
/* Als punt (X, Y) buiten de grenzen van het scherm ligt,
   roep dan 'fatal' aan, druk de incorrecte coordinaten af
   en stop.
*/
{ if (X < Ø || X > X__max || Y < Ø || Y > Y__max)
  { fatal();
    printf(
"Punt buiten scherm (X en Y zijn pixelcoordinaten):\n");
    printf("X = %d        Y = %d\n", X, Y);
    printf("x = %10.3f   y = %10.3f\n",
           X/horfact, (Y__max-Y)/vertfact);
    printf(
    "X__max = %d  Y__max = %d  x_max = %f y_max = %f\n",
           X__max, Y__max, x_max, y_max);
    printf("horfact = %f  vertfact = % f\n",
           horfact, vertfact);
    exit(1);
  }
}

void move(float x, float y)
/* Verplaats het huidige punt naar (x, y);
   x en y zijn schermcoordinaten.
*/
{ int XX, YY;
  XX = IX(x); YY = IY(y); check(XX, YY);
  if (fplot != NULL && (XX != X1 || YY != Y1))
    fprintf(fplot, "%6.3f %6.3f Ø\n", x, y);
```

```
    X1 = XX; Y1 = YY;
  }

  void draw(float x, float y)
  /* Trek een lijn van het huidige punt naar  (x, y). */
  { int X2, Y2;
    X2 = IX(x); Y2 = IY(y); check(X2, Y2);
    draw_line(X1, Y1, X2, Y2);
    if (fplot != NULL)
    fprintf(fplot, "%6.3f %6.3f 1\n", x, y);
    X1 = X2; Y1 = Y2;
  }


  int pixlit(int X, int Y)
  /* Informeer of pixel (X, Y) verlicht is. */
  { return (getpixel(X, Y) == foregroundcol);
  }


  static void prchar(char ch)
  /* Zend ch naar de parallelle printerpoort LPT1.        */
  { regs.x.dx=0;          /* Printerselectie              */
    regs.h.ah=0;          /* Zend byte van AL naar printer */
    regs.h.al=ch;         /* Byte bestemd voor de printer  */
    int86(0x17, &regs, &regs);
  }

  void printgr(int Xlo, int Xhi, int Ylo, int Yhi)
  /* Print de inhoud van rechthoek op matrixprinter. */
  { int n1, n2, ncols, i, X, Y, val, Xhi1;
    char line[720];                    /* 720 >= X__max */
    regs.x.dx = 0;                     /* LPT1          */
    regs.h.ah = 2;  /* Service 2: Get Printer Status */
    int86(0x17, &regs, &regs);  /* Printer services  */
    if ((regs.h.ah & 8) == 8)   /* Printer status in AH */
    { if (in_textmode) printf("Tekstmode (printgr)"); else
      textXY(0, Y__max - 15, "Printer niet gereed");
      return;
    }
    prchar(27); prchar('1');   /* Line spacing 7/72 inch */
    for (i=Ylo; i<=Yhi; i+=7)
    { checkbreak(); /* Laat DOS testen op console break  */
      Xhi1 = Xlo-1;
      for (X=Xlo; X<=Xhi; X++)
      { val=0;
```

```
      for (Y=i; Y<i+7; Y++)
      { val <<= 1; val |= (Y>Yhi ? Ø : pixlit(X, Y));
      }
      line[X] = val;
      if (val) Xhi1 = X;
    }
    ncols = Xhi1-Xlo+1;
    if (ncols)
    { n1=ncols%256; n2=ncols/256;
      prchar(27); prchar('L'); prchar(n1); prchar(n2);
      for (X=Xlo; X<=Xhi1; X++) prchar(line[X]);
    }
    prchar('\n');
  }
  prchar(27); prchar('@');
}


void setprdim(void)
/* Deze functie geeft zodanige waarden aan x_max en y_max
   dat de maten op de printer goed uitkomen.
*/
{ extern float x_max, y_max;
  extern int X__max, Y__max;
  setgrcon(iscolor());
  x_max = (X__max + 1)/12Ø.Ø; y_max=(Y__max + 1)/72.Ø;
}

 void textXY(int X, int Y, char *str)
 /* Geef in grafische mode string str op het scherm
    weer, beginnend op positie (X, Y).
 */
 { int height, width;
   height = textheight(str); width = textwidth(str);
   setviewport(X, Y, X+width, Y+height, Ø);
   clearviewport();  /* Wis eventuele oude tekst uit */
   outtext(str);
   setviewport(Ø, Ø, X__max, Y__max, Ø);
 }


 void text(char *str)
 /* Geef string str weer, beginnend in
    het 'huidige punt' (X1, Y1).          */
 { textXY(X1, Y1, str);
   X1 += strlen(str) * 8;
```

```
  check(X1, Y1);
}

void imove(int X, int Y)
/* Laat (X, Y) het nieuwe huidige punt zijn */
{ check(X, Y); X1=X; Y1=Y;
}


void idraw(int X, int Y)
/* Trek een lijnstuk van het huidige punt naar (X, Y)  */
{ check(X, Y);
  draw_line(X1, Y1, X, Y);
  X1=X; Y1=Y;
}


void wrscr(int row, int col, char *str)  /* In tekstmode */
/*  Schrijf, in tekstmode, karakters in het schermgeheugen.
    (row = Ø, 1, ..., 24;  col = Ø, 1, ..., 79)
*/
{ gotoxy(col+1, row+1);
  cputs(str);
}

void clearscr(void)
/* Te gebruiken in tekstmode.
   Verwar deze functie niet met clearpage, die in
   grafische mode wordt gebruikt.
*/
{ clrscr();
}

void settxtcursor(int row, int col)
/* Zet, in tekstmode, de cursor op de gewenste plaats. */
{ gotoxy(col+1, row+1);
}

void far * far _graphgetmem(unsigned size)
{ char *p;
  p = farmalloc((long)size);
  if (p == NULL) error("Geheugenruimte in _graphgetmem");
  return p;
}
```

```
void far _graphfreemem(void far *ptr, unsigned size)
{ farfree(ptr);
}
```

Als u GRPACK1 wilt gaan gebruiken is het belangrijk aandacht te besteden aan het derde argument van de functie *initgraph*, die aangeroepen wordt in *initgr.* Dit argument, in het handboek *pathtodriver* genoemd, moet vertellen waar in uw systeem de grafische driver te vinden is. Deze grafische drivers zijn files met een naam die eindigt op *.BGI.* Zo is die naam bijvoorbeeld *HERC.BGI* voor de Hercules Graphics Adapter. Deze grafische drivers worden geleverd op de diskettes die de Turbo C compiler, versie 1.5, bevatten. Als zij op uw systeem geplaatst zijn in de directory \ *turboc* van de 'current drive', dan kunt u *"\\ turboc"* schrijven, zoals in bovenstaande programmatekst gedaan is. De dubbele 'backslash' (\\) is hier nodig omdat alleen het karakterpaar \t, in overeenstemming met de taalregels van C, 'horizontal tab' zou betekenen. Als de grafische driver in kwestie niet in de opgegeven directory te vinden is, dan kijkt *initgraph* in de 'current directory'. U kunt in plaats van de echte directory ook de lege string "" opgeven; in dat geval moet de grafische driver zich in de current directory bevinden.

De laatste twee functies, *_graphgetmem* en *_graphfreemem*, vervangen de 'default'-functies die door de Turbo C functie *initgraph* (zie *initgr*) worden gebruikt om geheugenruimte voor de grafische driver aan te vragen. De vreemd genoteerde eerste regels van deze functies (met meer dan eens het woord *far* erin) zijn overgenomen uit de Turbo C versie 1.5 manual *Additions & Enhancements*. In connectie met een hoofdprogramma zoals D3D, dat zelf grote hoeveelheden geheugenruimte dynamisch aanvraagt, is het echt nodig deze twee functies zelf te leveren, omdat de overeenkomstige default-functies *malloc* in plaats van *farmalloc* gebruiken. (Toen ik dit aanvankelijk naliet, liep D3D in sommige gevallen gewoon vast; u kunt zich misschien voorstellen dat het mij buitengewoon veel moeite heeft gekost de oorzaak daarvan op te sporen!)

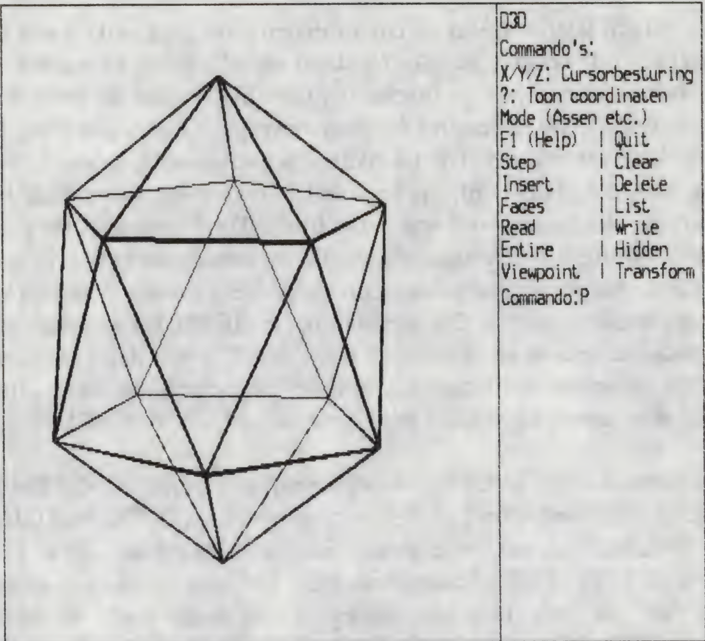
GRPACK1 is niet volledig equivalent met GRPACK. Bij gebruik van GRPACK hangt het effect van de functies *draw, idraw, draw*_line en *dot* af van de variabele *drawmode*. Zoals we hebben besproken, is de defaultwaarde van deze variabele gelijk aan 1. Maken we hem gelijk aan -1, dan maken genoemde functies de pixels donker en met *drawmode* = 0 inverteren ze de pixels. Bij GRPACK1 is dit alleen van toepassing op de functie *dot*, niet op de functies die lijnen trekken. Dit komt omdat in *draw*_line de nieuwe functie *line* van Turbo C wordt gebruikt, die niet de mogelijkheid biedt alle pixels van de lijn in kwestie te inverteren. Het inverteren van pixels (met behulp van *drawmode* = 0) wordt in D3D toegepast voor de cursor en voor de loodlijnen die bij cursorbewerkingen vanuit de cursor op het xy-vlak worden neergelaten. Let wel, we mogen niet eenvoudig alle pixels waaruit de cursor bestaat eerst verlichten en ze later donker maken, want dan zouden we andere lijnen uitpoetsen als we de cursor er overheen bewogen. Door de pixeltoestand tweemaal

te inverteren (één keer wanneer de cursor in een punt aankomt en één keer als hij het verlaat) blijft de oorspronkelijke toestand van alle pixels behouden. Nu wordt gelukkig voor de cursor zelf de functie *dot* gebruikt; omdat we (met de Turbo C functie *getpixel*) de pixeltoestand kunnen opvragen, kunnen we de (verlichte) 'voorgrondkleur' veranderen in de (donkere) 'achtergrondkleur' en vice versa. Met de functie *line*, die zoals eerder gezegd het karakter van een 'black box' heeft, kunnen we dit niet, maar voor wat programma D3D betreft is dit geen ernstig probleem. Tenslotte zijn de verticale lijnen die de cursor met het xy-vlak verbinden niet essentieel, dus in plaats van een meer complexe oplossing te vinden kunnen we die lijnen eenvoudig weglaten. Dit verklaart dat in GRPACK1 de functie *draw*_line alleen lijnstukken tekent als *drawmode* gelijk aan 1 is. Bij dezelfde versie van de module D3D worden de bedoelde verticale projectielijnen weggelaten als we GRPACK1 gebruiken en getekend (en later uitgewist) als we GRPACK gebruiken.

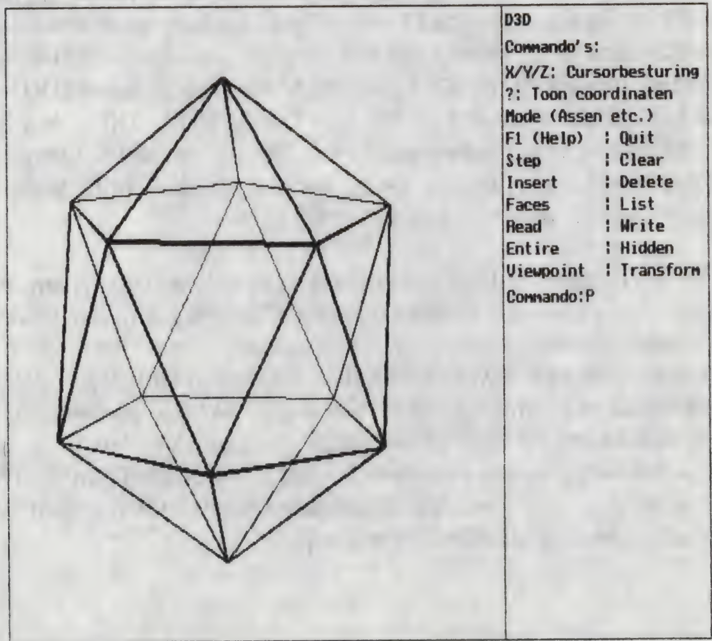
De programmatekst van GRPACK is omvangrijker dan die van GRPACK1, maar deze vergelijking is misleidend. Ik heb voor programma D3D zowel GRPACK als GRPACK1 gebruikt, wat resulteerde in de uitvoerbare files D3D.EXE, respectievelijk D3D1.EXE. Laatstgenoemde file was 11630 bytes groter dan eerstgenoemde en zelfs deze vergelijking is nog misleidend. We moeten niet vergeten dat D3D1.EXE tijdens executie de grafische driver laadt, wat D3D.EXE niet doet. Dit betekent dat voor de Hercules Graphics Adapter de grafische functies van Turbo C tijdens de uitvoering van het programma nog eens 10046 bytes extra gebruiken. Een versie D3D1.EXE (voor HGA) kost dus in totaal 21KB meer dan een versie D3D.EXE die werkt op HGA, CGA en EGA. Dit is de prijs die we moeten betalen voor de grafische routines van Turbo C versie 1.5. Om misverstand te vermijden haast ik mij hieraan toe te voegen dat deze prijs, gezien de vele mogelijkheden van die routines, niet onredelijk is.

Als we D3D.EXE en D3D1.EXE experimenteel met elkaar vergelijken, waarbij we gebruik maken van een betrekkelijk langzame 8088-processor, dan vinden we dat sommige bewerkingen, zoals het schoonmaken van het scherm, door eerstgenoemde sneller plaatsvinden dan door laatstgenoemde. Een ander verschil is het text-font dat in de grafische mode wordt gebruikt. De grafische routines van Turbo C gebruiken een (default-) karaktergrootte van 8 x 8, terwijl die grootte in GRPACK, zoals besproken in mijn boek *CGIP*, 11 x 8 is. De figuren 4.1(a) en (b) tonen het verschil in font (waarbij ik eerlijkheidshalve moet zeggen dat ik het tweede lettertype mooier vind dan het eerste.).

(a)



(b)



Figuur 4.1 Verschil in lettertype

(a) Font gedefinieerd in GRPACK
(b) Font (default) van Turbo C, verkregen door gebruikt te maken van GRPACK1

4.4 ELIMINATIE VAN VERBORGEN LIJNEN

Van alle D3D-commando's is *H*, waarmee we verborgen lijnen uit het beeld verwijderen, misschien wel het meest interessante. Een groot deel van mijn boek *Programming Principles in Computer Graphics* (*PPCG*) is aan de eliminatie van verborgen lijnen gewijd; zo'n uitvoerige discussie zou hier niet op zijn plaats zijn. Toch zullen we hier enige essentiële aspecten van de gebruikte algoritme bespreken, met inbegrip van de gebruikte datastructuren. Die algoritme is in essentie dezelfde als die welke gebruikt is in programma HIDLINPIX, dat te vinden is in *PPCG*. (Dat programma is werkelijk nuttig gebleken, zoals gedemonstreerd werd door bijvoorbeeld W.D. May in zijn artikel *3-D Images from Contour Maps*, in Dr. Dobb's Journal of Software Tools, november 1987.)

In plaats van dat programma zullen we nu een functie gebruiken die gedefinieerd is in de module HLPFUN en die aangeroepen wordt wanneer we commando *H* intypen. Vergeleken met HIDLINPIX biedt HLPFUN enkele verbeteringen en nieuwe mogelijkheden:

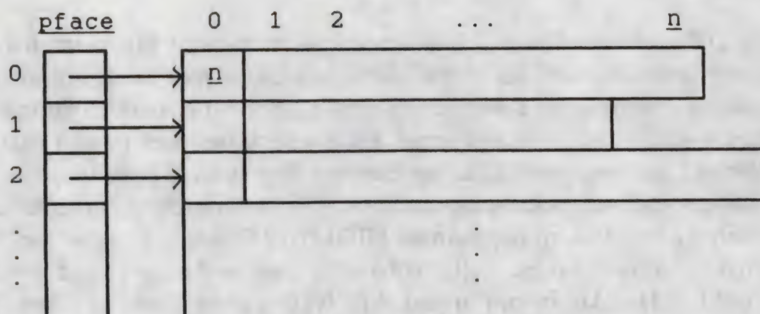
1 Er is een speciale behandeling van gekromde oppervlakken mogelijk, zoals we die aan het eind van paragraaf 1.5 hebben leren kennen.

2 Er wordt op een zodanige manier geheugenruimte aangevraagd dat rekening wordt gehouden met de ruimte die nog beschikbaar is.

3 Om eventuele problemen in verband met de stackgrootte te vermijden is recursie geëlimineerd.

4 De methode om een veelhoek in driehoeken te verdelen is verbeterd en werkt nu voor iedere veelhoek.

In de volgende bespreking zullen vaak namen van programmavariabelen worden genoemd. Dit geeft u de mogelijkheid ze op te zoeken, maar aangezien de programmamodule HLPFUN nogal complex is, kunt u dit misschien beter pas doen na deze toelichting gelezen te hebben.
De veelhoeken die de gebruiker door middel van commando *F* heeft opgegeven leveren ons zowel de lijnstukken die kandidaat zijn om te worden getekend als de grensvlakken die ze (geheel of gedeeltelijk) kunnen verbergen. De hoekpuntnummers van elke veelhoek worden opgeborgen in een aaneengesloten geheugengebied waartoe we toegang hebben via een pointer opgeborgen in 'array' *pface*. Eigenlijk is *pface* zelf ook een pointer variabele, waaraan we het beginadres van een dynamisch gereserveerd geheugengebied toekennen. Figuur 4.2 laat dit zien; we kunnen *pface* gebruiken als een array waarvan de elementen pointers naar integers zijn. (In C kan een pointer naar een integer in feite een pointer naar een rij integers zijn.)



Figuur 4.2 Datastructuur voor grensvlakken

We kunnen naar keuze **pface*[*i*] of *pface*[*i*][0] schrijven voor de integer waarnaar gewezen wordt door array-element *pface*[*i*]. Deze integer is gelijk aan het aantal puntnummers die volgen. Dus als *pface*[*j*][0] = *n*, dan zijn *pface*[*j*][1] t/m *pface*[*j*][*n*] de hoekpuntnummers van de *j*de veelhoek. Als we het nummer *i* van een punt weten, dan kunnen we de oogcoördinaten ervan vinden in de structuur *VERTEX*[*i*]. (Eigenlijk is *VERTEX* een pointervariabele, waaraan we in module D3D de waarde van variabele *p* toekennen.)

De zijden van de veelhoeken zijn de lijnstukken die we, voor zover ze zichtbaar zijn, moeten tekenen. Terwille van de eenvoud ontbinden we alle veelhoeken in driehoeken, die we daarna gaan gebruiken om uit te zoeken wat zichtbaar is en wat niet. Om dit uitzoekwerk voor een gegeven lijnstuk uit te voeren zouden we het lijnstuk kunnen combineren met alle driehoeken en daarbij bepalen of zekere delen ervan door geen enkele driehoek worden verborgen; zulke delen (die kunnen variëren van helemaal niets tot alles) moeten dan worden getekend. Dit zou evenwel leiden tot enorme rekentijden. Als we bijvoorbeeld 1000 lijnstukken en 1000 driehoeken hebben, dan zouden er 1000000 verschillende paren zijn, elk bestaande uit een lijnstuk en een driehoek. Mijn boek *PPCG* bespreekt een methode om dit aantal paren te reduceren en we zullen deze methode ook hier gebruiken. Wanneer we bezig zijn met een gegeven lijnstuk, dan hebben we een middel nodig om een redelijk beperkte deelverzameling van de verzameling van alle driehoeken op te bouwen en we willen daarbij in dit stadium niet eens alle driehoeken *inspecteren*. De sleutel tot de oplossing ligt in het scherm, waarop zowel lijnstukken als driehoeken worden geprojecteerd. Laten we, om de methode in eenvoudige bewoordingen duidelijk te maken, de kleinste viewport waarin het hele beeld past beschouwen als een schaakbord met 64 velden. Het idee is dat we in ieder veld informatie opbergen betreffende de driehoeken met een perspectivisch beeld dat punten bevat die in dat veld liggen. Laten we, om dit te verduidelijken, de letters *A* t/m *H* gebruiken voor de acht kolommen (tellend van links naar rechts) en de nummers 1 t/m 8 voor de rijen (van onder naar boven). Als nu bijvoorbeeld een

beelddriehoek ligt in de velden *A*1, *B*1 en *B*2 dan bergen we die informatie op een of andere manier op in deze velden. Als we dan later bezig zijn met een lijnstuk, dan bepalen we ook weer in welke velden van het schaakbord het beeld ervan ligt. Als bijvoorbeeld met lijnstuk PQ alleen de velden *E*5, *D*6, *C*7 en *B*8 geassocieerd zijn, dan worden alleen de driehoeken geselecteerd waarvan informatie is opgeborgen in deze velden, wat betekent dat de zojuist genoemde driehoek, die in de linker benedenhoek van het schaakbord ligt, wordt genegeerd. Op deze manier kunnen we een betrekkelijk kleine verzameling driehoeken vormen die lijnstuk PQ zouden kunnen verbergen; zolang we met PQ bezig zijn negeren we alle andere driehoeken.

In werkelijkheid hoeven we zelfs nog minder driehoeken te beschouwen dan we zojuist hebben besproken. Als er, voor een gegeven veld, verscheidene driehoeken zijn die dat veld *volledig* bedekken (wat betekent dat het hele vierkant ligt binnen de beelden van deze driehoeken), dan kunnen we al deze driehoeken negeren, met uitzondering van die welke zich het dichtst bij het oogpunt bevindt.

Wat we hierboven *veld* hebben genoemd, heet in *PPCG* een (device-independent) pixel, vandaar de daar gebruikte programmanaam HIDLINPIX en de naam HLPFUN voor de hier besproken functiemodule, die van dat programma afgeleid is. Omdat we in dit boek het woord *pixel* in de gebruikelijke (machine-afhankelijke) betekenis hebben gebruikt, zou het verwarrend kunnen zijn als we het nu ook gingen gebruiken voor grotere delen van het scherm. Hoewel we ons hierboven vierkante velden van een schaakbord hebben voorgesteld, is zo'n deel van het scherm in het algemeen niet vierkant maar rechthoekig, zodat we in plaats van over 'veld' liever over *rechthoek* zullen spreken. Niet het gehele scherm, maar de kleinste rechthoek (met horizontale en verticale zijden) waarin het tweedimensionale beeld past zal worden verdeeld in $N \times N$ rechthoeken van gelijke grootte; in de programmaversie die in Appendix B is opgenomen wordt $N = 15$ gebruikt. De waarde van N is niet van invloed op het eindresultaat, dus die waarde moet niet worden verward met een soort 'resolutie'. Wel wordt de rekentijd erdoor beïnvloed. In enige experimenten bleek dat bij gebruik van de IBM PC de rekentijd gewoonlijk toeneemt als we de waarde 15 vervangen door een veel kleinere of een veel grotere waarde (zoals bijvoorbeeld 4 of 30).

We zullen de zojuist genoemde rechthoeken associëren met elementen van array *SCREEN*. Array-element *SCREEN*[*i*][*j*] (waarin i en j niet-negatieve gehele getallen kleiner dan N zijn) correspondeert met de rechthoek in de *i*de kolom en de *j*de rij, met andere woorden, we berekenen i uit X en j uit Y. Laten we eenvoudshalve het woord *rechthoek* ook gebruiken voor de elementen van array *SCREEN* zelf en zeggen dat we (informatie omtrent) driehoeken willen opbergen in rechthoeken.

Daar onze rechthoeken alle van dezelfde grootte zijn en iedere rechthoek geassocieerd moet worden met een variabel aantal driehoeken, zullen we de driehoeken in werkelijkheid buiten de rechthoeken opslaan, maar op een zodanige

manier dat de rechthoeken ons kunnen vertellen waar we ze kunnen vinden. We zullen gebruik maken van de tabel *TRIANGLE*, die een structuur (in de zin van de taal C) bevat voor iedere driehoek. In deze structuur vinden we zowel de hoekpuntnummers A, B, C van driehoek i als de coëfficiënten a, b, c, h van

$$ax + by + cz = h,$$

dat is de vergelijking van het vlak waarin de driehoek ligt. We kunnen dan een enkele integer i gebruiken om een driehoek te identificeren; we zouden daarom in iedere rechthoek een verzameling van driehoeknummers willen opbergen als dat mogelijk was. Omdat zo'n verzameling variabel van grootte is, bergen we in iedere rechthoek een startpointer van een lineaire lijst op en in plaats van een driehoeknummer in een rechthoek op te bergen stoppen we dat nummer in de lineaire lijst die in die rechthoek begint. Ik deed dit eerst op de meest natuurlijke manier, met echte pointers en geheugenruimte reserverend voor ieder element van de lijst, maar dit bleek een ernstig nadeel op te leveren. Omdat we deze methode moeten implementeren in functie *hlpfun*, die in programma D3D meer dan eens kan worden aangeroepen, moeten we niet vergeten alle dynamisch gereserveerde geheugenruimte ook weer vrij te geven voordat we naar het hoofdprogramma terugkeren, zodat dezelfde geheugenruimte later weer opnieuw gebruikt kan worden. Als het afgebeelde voorwerp ingewikkeld is kunnen de lineaire lijsten met driehoeknummers behoorlijk lang zijn en groot in aantal. Als gevolg hiervan bleek dat het vrijgeven van de geheugenruimte gebruikt voor lijstelementen in zulke gevallen enorm veel tijd kostte. De gemiddelde gebruiker zal er nog wel begrip voor kunnen opbrengen dat het verwijderen van verborgen lijnen uit een beeldfiguur wat rekentijd kost, maar hij (of zij) zal verbaasd zijn als de computer bijna evenveel tijd nodig heeft om van die beeldfiguur af te komen. Aangezien dit werkelijk het geval was, heb ik het programma gewijzigd en de lineaire lijst geïmplementeerd als een tabel, *TRLIST* geheten. We kunnen dan de hele tabel met lijstelementen in één enkele handeling vrijgeven. Iedere regel in de tabel *TRLIST* kan worden gebruikt als een element van een lineaire lijst; zo'n regel is een structuur met de volgende twee velden:

ptria Een integer die verwijst naareen plaats in de tabel *TRIANGLE*. Dus als deze integergelijk is aan i, dan is *TRIANGLE*[i] de driehoek in kwestie.

next Een integer die verwijst naar het volgende element van de lineaire lijst. (Als deze integer gelijk is aan $k \geqslant 0$, dan is *TRLIST*[k] dat volgende element. Als k gelijk is aan -1, dan is er geen volgend element.) Er wordt naar het eerste element van iedere lijst verwezen door een getal dat in array *SCREEN* is opgeborgen, zoals we hieronder zullen zien.

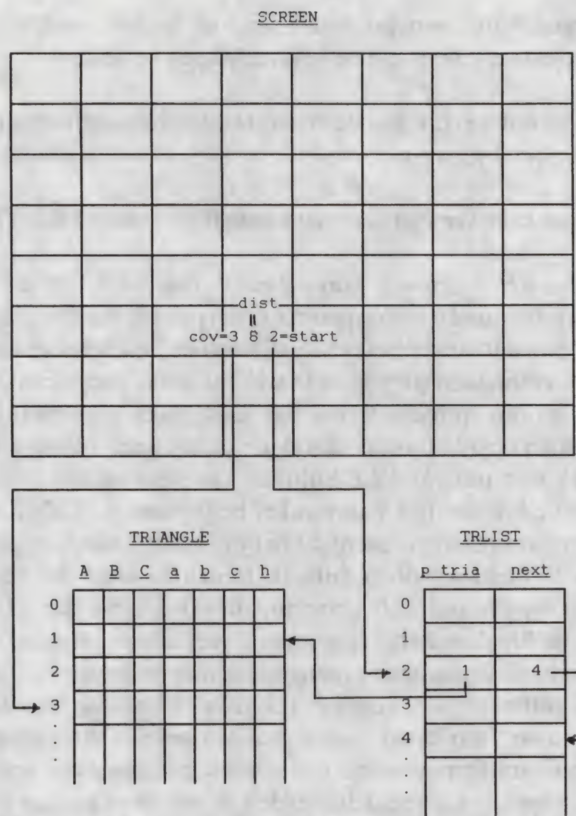
Iedere 'rechthoek' *SCREEN*[i][j] van het scherm is een structuur met de volgende drie velden:

tr_cov Een integer *i* die verwijst naar een driehoek: *TRIANGLE*[*i*] is de dichtstbijzijnde driehoek die de hele rechthoek bedekt.

tr_dist Een *float*-waarde, gelijk aan de afstand tussen het oogpunt en de driehoek waarnaar *tr*_cov verwijst.

start Een integer *k* die verwijst naar het eerste lijstelement *TRLIST*[*k*].

In figuur 4.3 hebben we *tr*_cov = 3, *start* = 2 en *tr*_dist = 8.0. Het laatste betekent dat er een afstand 8.0 is tussen het oogpunt en een punt P van driehoek 3, zodanig dat de projectie P' van dat punt P het centrum is van de beschouwde rechthoek. We hebben hier 8 x 8 rechthoeken (terwijl er 15 x 15 zijn in het programma). Figuur 4.3 toont een situatie die kan optreden terwijl, op basis van de gegeven driehoeken, de lineaire lijsten worden opgebouwd. Nadat deze 'eerste scan' voltooid is, wordt een eventuele driehoek waarnaar *SCREEN*[*i*][*j*].*tr*_cov verwijst ook nog toegevoegd aan het begin van de lineaire lijst waarvan het beginpunt in *SCREEN*[*i*][*j*].*start* is opgeborgen. In ons voorbeeld betekent dit dat driehoek 3 wordt opgenomen in de lineaire lijst van de beschouwde rechthoek (tenzij er voor die rechthoek een driehoek volgt met een afstand tot het oogpunt die nog kleiner dan 8.0 is).
Wanneer we met veelhoeken bezig zijn, komen niet alleen driehoeken maar ook lijnstukken beschikbaar. Zij moeten ergens worden opgeborgen, zodat we ze later, voor zover ze zichtbaar zijn, kunnen tekenen. Maar we moeten dat niet onvoorwaardelijk doen. Ten eerste zult u zich herinneren van paragraaf 1.7 (zie figuur 1.17) dat hoekpuntnummers met een minteken kunnen zijn opgeborgen, als codering voor kunstmatig aangebrachte zijden in veelhoeken met gaten. Het is duidelijk dat we zulke zijden eenvoudig zullen negeren. Aangezien dit een uiterst eenvoudige aangelegenheid is zullen er niet meer over praten en onze discussie beperken tot positieve puntnummers. Ten tweede moeten we ons realiseren dat een ribbe van een ruimtelijk lichaam de snijlijn van twee grensvlakken is, wat betekent dat die ribbe in onze rekenmethode twee keer zal optreden; omdat hij ten hoogste één keer getekend hoeft te worden kunnen we hem beter de tweede keer negeren. Deze twee aspecten zijn niet nieuw; zij worden ook besproken in *PPCG*. Er is een derde voorwaarde die we aan elk eventueel te tekenen lijnstuk stellen, die nieuw is in dit boek. Zoals we weten voorziet D3D in een speciale behandeling van gekromde oppervlakken: als de hoek tussen de normaalvectoren van twee snijdende grensvlakken kleiner is dan een zekere, door de gebruiker opgegeven, drempelwaarde, dan moet de snijlijn van die twee grensvlakken niet worden getekend. Dit betekent dat voor iedere ribbe die als zijde van een veelhoek beschikbaar komt en die we willen opbergen, we niet alleen de nummers P en Q van de eindpunten van de ribbe, maar ook de drie componenten *a*, *b* en *c* van een normaalvector moeten opbergen.
Let erop dat dit de coëfficiënten van *x, y* en *z* zijn in de vergelijking van het vlak waarin de veelhoek ligt.



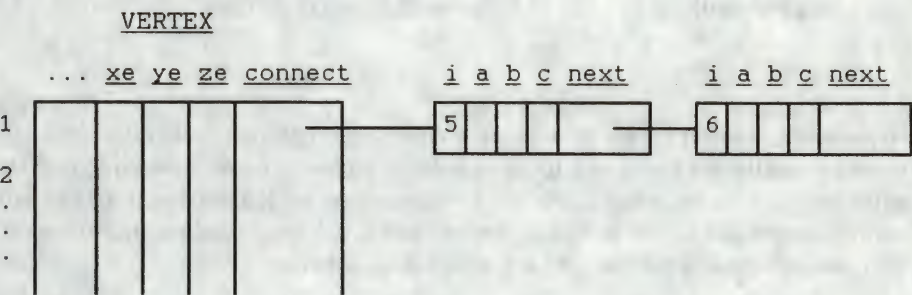
Figuur 4.3 Scherm, lineaire lijsten en tabel met driehoeken

Op deze manier kunnen we de normaalvector [*a b c*] later gebruiken als hetzelfde lijnstuk opnieuw verschijnt, maar nu als zijde van een andere veelhoek, waarvan eveneens een normaalvector is gegeven.

Er zijn enkele snijlijnen die, als ze zichtbaar zijn, getekend moeten worden, zelfs als de hoek tussen de bovenbedoelde twee normaalvectoren kleiner is dan de gegeven drempelwaarde. Dit zijn de *contouren* van het voorwerp. Zij zijn te zien in bijvoorbeeld de cilinder van figuur 2.6. De verticale contourlijnen uiterst links en rechts moeten worden getekend zelfs al ontstaan ze als ribbe op dezelfde manier als andere, die in de tekening worden weggelaten. Gelukkig hoeven we geen bijzondere maatregelen te nemen om die contourlijnen te tekenen. De reden is dat contourlijnen snijlijnen zijn van een zichtbaar vlak en een *achtervlak* (backface) en daarom maar één keer zullen optreden. Hoewel nog niet vermeld in onze discussie, negeren we veelhoeken die achtervlakken zijn: ten eerste wordt alles wat zij voor het oog verbergen ook verborgen door zichtbare grensvlakken en ten tweede is elke zijde van zo'n veelhoek òf verborgen òf tevens zijde van een zichtbare veelhoek.

Zoals u zich herinnert moet de gebruiker de hoekpunten van elke veelhoek in antikloksgewijze volgorde opgeven en moet het tweede hoekpunt convex zijn. We kunnen dan na projectie een achtervlak onderscheiden van een (mogelijk) zichtbaar vlak door gebruik te maken van de oriëntatie van de eerste drie hoekpuntnummers. Als de eerste drie punten na projectie niet meer anti-kloksgewijs georiënteerd zijn dan is het grensvlak in kwestie een achtervlak. Het komt nu goed uit dat iedere ribbe die contourlijn is maar één keer beschikbaar komt doordat het achtervlak waarin hij ligt niet bijdraagt aan de levering van lijnstukken.

Het zal nu duidelijk zijn dat iedere ribbe PQ die kandidaat is om te worden opgeborgen eerst moet worden opgezocht om te zien of al eerder een kopie ervan is opgeborgen. Zo ja, dan zullen we het nieuwe exemplaar zeker niet opbergen, maar, afgezien hiervan, het kan zelfs zijn dat we het oude exemplaar moeten verwijderen (zodat het aantal opgeborgen lijnstukken met 1 afneemt). Natuurlijk zal dit laatste nooit gebeuren als de gebruiker een drempelwaarde 0 opgegeven heeft.
Om lijnstukken efficiënt te kunnen opzoeken, bergen we ze op in lineaire lijsten, zoals afgebeeld in figuur 4.4. Voor elke ribbe PQ verwisselen we eventueel (de numerieke waarden van) P en Q zodanig dat hierna P kleiner is dan Q. Puntnummer Q wordt dan opgeborgen in een element van de lineaire lijst waarvan het startpunt ligt in *VERTEX*[*P*]. (Zoals in het begin van deze paragraaf vermeld, is *VERTEX* een tabel waarin de coördinaten van alle hoekpunten te vinden zijn.) Naast puntnummer Q bergen we ook de coëfficiënten *a*, *b* en *c* op die behoren bij het vlak waarin we juist lijnstuk PQ hebben gevonden, zodat we de 'drempeltest' kunnen uitvoeren wanneer later een kopie van PQ beschikbaar komt. Figuur 4.4 laat zien hoe de lijnstukken (1, 5) en (1, 6) worden opgeborgen.



Figuur 4.4 Lineaire lijst voor lijnstukken

Voor verdere bijzonderheden verwijs ik naar mijn boek *PPCG* en naar de tekst van de programmamodule HLPFUN, die opgenomen is in Appendix B.

4.5 EEN HULPPROGRAMMA VOOR HP-PLOTTERS

Er zijn twee redenen waarom onze grafische uitvoer op de matrixprinter niet bepaald van hoge kwaliteit is. Ten eerste geeft de printer geen erg goed contrast tussen zwart en wit en ten tweede hebben we de resolutie van het beeldscherm gebruikt voor de printer, wat soms leidt tot nogal 'kartelige' lijnen, vooral als CGA, met een resolutie van 640 x 200, wordt gebruikt. Onze nogal complexe driedimensionale berekeningen en in het bijzonder onze algoritme voor verborgen lijnen leveren ons de exacte eindpunten van ieder lijnstuk in het tweedimensionale resultaat. Omdat we mogelijk gebruik kunnen maken van grafische uitvoerapparatuur die de berekende tweedimensionale resultaten kan verwerken biedt D3D de mogelijkheid deze resultaten in een file te schrijven. Zoals vermeld in paragraaf 1.5, typen we hiertoe, na een van de commando's *H* of *E*, de letter *O* in (als afkorting van 'Output file'). Dit type file moet goed worden onderscheiden van de files die we *objectfiles* hebben genoemd. Laten we consequent een naam eindigend op *.DAT* gebruiken voor objectfiles, die als invoer voor D3D dienen; de uitvoerfiles verkregen met commando *O* hebben altijd een naam die eindigt op *.PLT*. Als D3D bijvoorbeeld de objectfile *CUBE.DAT* heeft gelezen (of heeft geschreven ten gevolge van commando *W*) dan zal ons commando *O* de file *CUBE.PLT* laten ontstaan. Deze naam zal ook gekozen worden als de naam van uw objectfile begint met *CUBE*, maar, tegen mijn advies, een andere extensie dan *.DAT* of helemaal geen extensie gekregen heeft. Als er nog geen filenaam bekend is omdat u noch een invoerfile noch commando *W* heeft gebruikt, dan zal commando *O* de uitvoerfile *NONAME.PLT* doen ontstaan.

We duiden een uitvoerfile van D3D, verkregen met commando *O* (en met een naam eindigend op *.PLT*), aan met de term *plotfile*. Plotfiles hebben een bijzonder eenvoudig formaat: het zijn ASCII-files waarin alleen maar regels van de vorm

x y code

voorkomen, waarbij x en y tweedimensionale coördinaten zijn die respectievelijk variëren van 0 tot 10 en van 0 tot 7. Het derde element, *code*, is hetzij 0, met de betekenis *pen hoog*, hetzij 1, met de betekenis *pen laag*. De oorsprong van het coördinatenstelsel ligt in de linker benedenhoek van een denkbeeldig vel papier. We kunnen bijvoorbeeld als volgt een rechte hoek tekenen:

```
2.000  6.000   0
2.000  1.000   1
5.500  1.000   1
```

We stellen ons een pen voor die eerst wordt *verplaatst* naar het punt met de coördinaten $x=2$, $y=6$. Daarna *trekt* hij een lijn naar punt $x=2$, $y=1$ en vandaar naar punt $x=5.5$, $y=1$. De twee penposities 'hoog' $=0$ en 'laag' $=1$ tijdens

penbewegingen kunnen we ook onderscheiden door de hierboven gebruikte werkwoorden 'verplaatsen' en 'trekken', ofwel, in het Engels, *move* en *draw*. Dus we kunnen bovenstaande drie regels symbolisch weergeven door:

```
move(2.0, 6.0);
draw(2.0, 1.0);
draw(5.5, 1.0);
```

Daar *move* en *draw* echte C-functies zijn, die we hebben besproken in paragraaf 4.3, mogen we verwachten dat het maken van een programma voor de conversie van een plotfile in een echte tekening niet al te moeilijk zal zijn. Mijn eerste versie van dit programma was alleen bedoeld om uitvoer op een Hewlett-Packard plotter te leveren, maar ik heb het daarna nog iets uitgebreid, waardoor we er ook uitvoer op het beeldscherm en op de printer mee kunnen verkrijgen. U kunt het programma starten door de regel

```
PLOTHP
```

in te typen. Er verschijnt dan de regel

```
Naam van invoerfile:
```

op het scherm, wat betekent dat we de naam van een plotfile, zoals bijvoorbeeld *CUBE.PLT* moeten intypen. Als deze file werkelijk bestaat verschijnen de volgende vragen:

```
Uitvoer op beeldscherm? (J/N)
Uitvoer op matrixprinter? (J/N)
Uitvoer op HP-plotter? (J/N)
```

Dus zelfs als er geen HP-plotter beschikbaar is kan PLOTHP nuttige diensten bewijzen. Hoewel we al eerder, namelijk bij het gebruik van D3D, in staat zijn gebleken grafische uitvoer op het beeldscherm en op de matrixprinter te verkrijgen, kan het wenselijk zijn dit later te herhalen, bijvoorbeeld wanneer we de printer van een nieuw lint hebben voorzien; als het afgebeelde voorwerp ingewikkeld is zal programma PLOTHP het veel sneller afbeelden dan D3D dat doet, omdat eerstgenoemd programma geen enkele driedimensionale berekening hoeft uit te voeren. Als ons antwoord op de eerste vraag *J* is, dan wordt om een schaalfactor gevraagd. Als we het beeld bijvoorbeeld op de helft van de normale grootte willen afbeelden dan moeten we 0.5 intypen; in de meeste gevallen zullen we de schaalfactor 1 gebruiken. (Dit in scherpe tegenstelling tot schaalfactoren voor plotters, zoals we hierna zullen zien.) Geschaald met de gegeven factor zal het beeld nu op het scherm verschijnen, evenals, als we daarom gevraagd hebben, op de printer.

Als u zo gelukkig bent van een HP-plotter gebruik te kunnen maken, dan kunt u de meest interessante mogelijkheid van PLOTHP benutten door *J* te antwoorden op de laatste van de bovenstaande drie vragen. De plotter moet verbonden zijn met een seriële I/O-poort, in de MS-DOS-terminologie *COM1* of *COM2*. Er is een aantal parameters die we de eerste keer dat we plotten moeten opgeven:

```
Nummer van de seriele poort (1 voor COM1, 2 voor COM2):
Baud-rate
(110, 150, 300, 600, 1200, 2400, 4800, 9600):
Indien onbekend, probeer de eerstgenoemde mogelijkheid:
Pariteit (0 = geen, 1 = oneven, 2 = even):
Aantal stopbits? (1 of 2):
Aantal bits per karakter (8 of 7):
```

Na elk van de zes bovenstaande dubbele punten dient u een van de gesuggereerde antwoorden in te typen. Het beste kunt u die opzoeken in de technische documentatie van uw computer en uw plotter. U vindt misschien een verwijzing naar DIP-schakelaars op de plotter, die bijvoorbeeld de baud-rate vastleggen. Op de plotters die ik heb gebruikt moest ik de laatste drie vragen respectievelijk met 0, 1 en 8 beantwoorden, dus u kunt dat ook eerst eens proberen als u het moeilijk vindt de juiste instellingen in de handboeken op te zoeken. Het zou vervelend zijn als we elke keer dat we PLOTHP uitvoeren deze zes vragen moesten beantwoorden; het leek mij daarom een goed idee PLOTHP de antwoorden te laten schrijven in een 'configuratiefile' en ik heb dit idee als volgt geïmplementeerd. Programma PLOTHP kijkt of er in de 'current directory' een file bestaat met de naam PLCONFIG.TXT. Aangezien dit de eerste keer dat u het programma draait niet het geval zal zijn, zal om de genoemde zes parameters worden gevraagd. Het programma creëert dan de file PLCONFIG.TXT en de parameters die u ingetypt heeft worden daarin geschreven. Als u dan later het programma weer uitvoert zal het programma die file vinden; het leest dan de parameters uit de file. Om u in staat te stellen ze te veranderen, worden ze getoond op het scherm met de vraag of ze correct zijn. Zo niet, dan kunt (en moet) u er nieuwe waarden voor intypen.
Nadat u de parameters voor de seriële poort heeft ingetypt wordt om een schaalfactor gevraagd die van toepassing is op de coördinaten die van de invoerfile worden gelezen. Zoals bekend variëren de x-coördinaten van 0 tot 10 en de y-coördinaten van 0 tot 7. In plaats hiervan heeft de plotter heel grote geheeltallige waarden nodig, die variëren tussen bijvoorbeeld 0 en 11420. Laatstgenoemd getal is de grootste x-waarde die mogelijk is op een HP7225A/B-plotter. Het wordt uitgedrukt in plottereenheden, waarbij één plottereenheid 0.025 mm is. De plotter zelf biedt mogelijkheden voor schaling, maar omdat deze bewerking buitengewoon eenvoudig is doet PLOTHP dit zelf, gebruik makend van de factor die we als gebruiker hebben opgegeven. We hoeven onze tekening niet te baseren op de mechanische grenzen van de plotter; we kunnen bijvoorbeeld 8000 (in plaats van 11420) kiezen als grootste x-coördinaat, wat betekent dat we een schaalfactor van

800 moeten opgeven. Het gevolg van dit laatste is dat de tekening een lengte van 800 x 10 = 8000 plottereenheden heeft, wat gelijk is aan 8000 x 0.025 = 200 mm. De grootte in de y-richting wordt dan 800 x 7 x 0.025 = 140 mm. Bij praktisch gebruik zult u al gauw zien wat voor uw toepassing de beste schaalfactor is. Het belangrijkste wat u dient te onthouden is dat de waarde ervan behoorlijk groot (zoals bijvoorbeeld 800) moet zijn.
Tenslotte dient ook vermeld te worden dat we de snelheid, die de pen tijdens het trekken van lijnen heeft, kunnen wijzigen. In plaats van de automatisch gekozen maximale snelheid, die bij de door mij gebruikte plotter 25 cm/s bedraagt, geeft een lagere waarde (tussen 1 en 25) betere resultaten omdat de inkt dan meer tijd heeft om het papier te bereiken. Als u de vraag

```
Standaard snelheid? (J/N):
```

negatief beantwoord, dan wordt gevraagd de gewenste snelheid op te geven en onmiddellijk hierna zal de plotter op basis van de gegeven plotfile gaan tekenen. Vele tekeningen in dit boek zijn op deze manier gemaakt en zoals we kunnen verwachten is de kwaliteit ervan veel beter dan die van een aantal andere illustraties, die met een matrixprinter verkregen zijn.
Bovenstaande discussie van PLOTHP is bedoeld voor de gebruiker, die niet over programmeerkennis hoeft te beschikken. We zullen nu enkele nogal technische programmadetails gaan bespreken. In het handboek van uw HP-plotter vindt u een uiteenzetting over de 'Hewlett-Packard Graphic Language' (HP-GL), een 'taal' die een veertigtal instructies bevat, waarvan wij er maar vijf gebruiken, namelijk:

```
IN      Initialize
VS      Velocity Select
PU      Pen Up
PD      Pen Down
PA      Plot Absolute
```

Als voorbeeld nemen we de volgende karakterstring:

```
IN;VS4;PU;PA4000,3000;PD;PA4000,5000;PU;
```

Wanneer we deze naar de plotter sturen dan worden achtereenvolgens de volgende stappen genomen:

1 IN initialiseert de plotter.
2 VS kiest een snelheid van 4 cm/s.
3 PU brengt de pen omhoog.
4 PA verplaatst de pen naar het punt met plottercoördinaten $X = 4000$, $Y = 3000$.
5 PD brengt de pen omlaag.

6 PA trekt de lijn van het punt genoemd in stap 4 naar punt $X=4000$, $Y=5000$.
7 PU brengt de pen omhoog.

Programma PLOTHP leest regels tekst van de gegeven plotfile en converteert ze in HP-GL-instructies; elk van deze wordt geplaatst in een string *str* en dan naar de seriële poort gezonden door middel van de functie-aanroep

```
ser_str(str);
```

Deze functie, *ser*_str, zendt alle karakters *ch* in de string *str* die aan het afsluitende nulkarakter voorafgaan naar de seriële poort door middel van

```
ser_char(ch);
```

Dit zijn twee zelfgemaakte functies; eerstgenoemde zal voor een ervaren C-programmeur geen moeilijkheden opleveren, maar de tweede vereist, zoals we zullen zien, enige kennis van datacommunucatie.
Er is een ROM-BIOS-call voor seriële (RS232) communicatie-faciliteiten, waarvan we gebruik kunnen maken door middel van interrupt 20 (hexadecimaal 0x14), zoals heel uitvoerig wordt uitgelegd in *The Peter Norton Programmer's Guide to the IBM PC*. Er zijn vier faciliteiten ('services'), genummerd 0, 1, 2, 3. Een assembleertaalprogrammeur zal, alvorens de software-interrupt uit te voeren, het faciliteitnummer in register AH plaatsen en, in geval van faciliteit 1, het over te zenden karakter in register AL zetten. Ook moet een code (0 voor COM1 en 1 voor COM2) voor de seriële poort in register DX worden geplaatst. Hoewel we met C werken en niet met assembleertaal, kunnen we dit allemaal gebruiken door de functie *int86* aan te roepen, die we in GRPACK ook voor andere doeleinden hebben gebruikt (zie Section 1.2.6 van mijn boek *CGIP* of paragraaf 4.3 van het onderhavige boek). Deze functie is beschikbaar in veel C-implementaties voor de IBM PC, inclusief Turbo C, Lattice C en Microsoft C. Hoewel ik *int86* in de eerste versie van PLOTHP heel bevredigend heb gebruikt, heb ik hem vervangen door de functie *bioscom*, die ik later in het handboek van Turbo C aantrof. Laatstgenoemde functie is voor ons doel gemakkelijker in het gebruik zodat programma PLOTHP er leesbaarder door wordt. De naam *bioscom* betekent waarschijnlijk 'communicatie door middel van BIOS': de functie kan alleen worden gebruikt voor seriële communicatie. In plaats van registers te gebruiken, zoals dat in assembleertaal gebeurt, kunnen we eenvoudig zeggen wat we willen door middel van functie-argumenten, zoals het functieprototype suggereert:

```
int bioscom(int cmd, char byte, int port);
```

Deze regel komt voor in de header-file *bios.h*, die we daarom met behulp van een 'include-regel' in ons programma zullen opnemen. Voor een volledige uitleg van

bioscom kunt u het beste het handboek van Turbo C raadplegen. Hier bespreken we deze functie alleen voor zover we hem nodig hebben. Het eerste argument, *cmd*, is bovengenoemd servicenummer:

0 Stel de communicatieparameters in volgens de waarde van *byte*.
1 Zend het karakter in *byte* naar de seriële poort.
2 Ontvang een karakter vanuit de seriële poort; het is te vinden in de acht minst significante bits van de teruggegeven functiewaarde.
3 Geef de huidige status van de seriële poort terug.

Als $cmd = 0$, dan zegt het tweede argument hoe de communicatieparameters moeten worden ingesteld. Dit argument, *byte*, moet geïnterpreteerd worden als een rij van acht bits, van links naar rechts genummerd 7, 6, ..., 0. Er zijn in *byte* vier groepen bits:

7, 6, 5: De baud-rate, gecodeerd als 000, 001, ..., 111 voor respectievelijk 110, 150, 300, 600, 1200, 2400, 4800, 9600 baud.

4, 3: 00, 01 of 11 als codes voor *Geen pariteit*, *Oneven pariteit* of *Even pariteit*.

2: 0 = *Eén stopbit*, 1 = *Twee stopbits*.

1, 0: 10 = *7 databits*, 11 = *8 databits*.

Het derde argument, *port*, is een code voor het poortnummer: het is 0 voor *COM1* en 1 voor *COM2*.

Hoewel dit misschien allemaal erg technisch in uw oren klinkt, is het letterlijk toepassen ervan niet moeilijk, mits de baud-rate, pariteit en aantallen stop- en databits bekend zijn. Toen ik dit deed werkte alles prima voor een paar eenvoudige testgevallen, maar het ging mis voor wat ingewikkelder tekeningen tengevolge van *buffer-overflow.* Ik had in het plotter-handboek een hoofdstuk over 'handshake protocols' gezien, dat mij meer geschikt leek voor elektrotechnici dan voor programmeurs. In een paragraaf getiteld 'Hardwire Handshake Mode' vond ik een stroomschema waarin een wachtlus voorkwam en hoewel de gebruikte technische termen mij niet allemaal even duidelijk waren, had ik er voldoende aan om de volgende oplossing te bedenken, die nog correct bleek te werken ook:

```
void ser_char(char ch)
{ int retval;
  do
  { retval = bioscom(3, byte, port);
```

```
   } while ((retval & 0x0030) != 0x0030);
   bioscom(1, ch, port);
}
```

Voordat karakter *ch* naar de seriële poort wordt gezonden (met gebruik van servicenummer 1), wacht de processor in de do-while-lus totdat hij dat mag doen. Binnen de lus vragen we naar de status van de seriële poort (gebruik makend van servicenummer 3). Functie *bioscom* geeft, als hij met servicenummer 3 wordt aangeroepen, de huidige poortstatus terug als functiewaarde. Elk bit van het teruggegeven gehele getal heeft een of andere betekenis die ongetwijfeld voor datacommunicatie-experts volkomen duidelijk zal zijn. Zoals volgt uit de hexadecimale constante 0*x*0030 in bovenstaand programmafragment, worden alleen de bits 5 en 4 geselecteerd; hun betekenis is respectievelijk *Data set ready* en *Clear to send*. Pas wanneer deze twee bits in de teruggegeven functiewaarde gelijk aan 1 zijn wordt de lus beëindigd. Op deze manier kunnen we 'overflow' van de databuffer in de plotter voorkomen. Verdere bijzonderheden zullen duidelijk worden aan de hand van de programmatekst zelf. Als u deze programmatekst zelf wilt laten compileren enzovoort, dan moet u eraan denken dat de header-file GRPACK.H beschikbaar moet zijn en dat GRPACK.OBJ aan de linker moet worden aangeboden.

```
/* PLOTHP:
      Het hoofddoel van dit programma is een invoerfile xxx.PLT
      om te zetten naar een tekening op een Hewlett-Packard-
      plotter. Bovendien kan het deze tekening op het beeld-
      scherm en op een matrixprinter produceren. De invoerfile
      moet bestaan uit regels van de vorm

         x  y  c

      De coordinaten x en y zijn niet-negatieve reele getallen,
      niet groter dan respectievelijk 10.0 en 7.0 en de code c
      is ofwel 0 (pen hoog) ofwel 1 (pen laag).
*/
#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <ctype.h>
#include <process.h>
#include "grpack.h"


void display(int printer);
void plotter(void);
```

```
void initplotter(void);
float scalingfactor(void);
void ser_str(char *p);
void ser_char(char ch);
void initialize(void);
char getcapital(void);

FILE *fp;
char filnam[40];
unsigned char byte;
int penposition, port;

main()
{ char ch;
  printf("Naam van invoerfile: "); scanf("%s", filnam);
  fp = fopen(filnam, "r");
  if (fp == NULL) {printf("File-probleem"); exit(1);}
  printf("Uitvoer op beeldscherm? (J/N): ");
  ch = getcapital();
  if (ch == 'J')
  { printf("\nUitvoer op matrixprinter? (J/N): ");
    ch = getcapital();
    display(ch == 'J');
    fclose(fp); fp = NULL;
  }
  printf("\nUitvoer op HP-plotter? (J/N): ");
  ch = getcapital();
  if (ch == 'J')
 { if (fp == NULL)
   { fp = fopen(filnam, "r");
     if (fp == NULL){printf("File-probleem"); exit(1);}
   }
   plotter();
 }
 exit(0); /* Sluit file fp, indien open */
}

void display(int printer)
{ float f, x, y, xfact, yfact, fact;
  char ch, code;
  f = scalingfactor();
  if (printer) setprdim();
     /* Wijzig x_max en y_max (float)  */
  initgr();         /* Definieer X__max en Y__max (int)     */
```

```
    xfact = f*x_max/10.0; yfact = f*y_max/7.0;
    fact = (xfact < yfact ? xfact : yfact);
    while (fscanf(fp, "%f %f %d", &x, &y, &code) == 3)
    { ch = getc(fp);
      if (code < 0 || code > 1 || ch != '\n')
      { textXY(40, 40, "Incorrect file-formaat"); endgr();
        exit(1);
      }
      x *= fact; y *= fact;
      if (x > x_max || y > y_max)
      { to_text();
        printf("Buiten de grenzen: x = %6.3f  y = %6.3f\n",
                x, y);
        return;
      }
      if (code) draw(x, y); else move(x, y);
    }
    if (printer) printgr(0, X__max, 0, Y__max);
    endgr();
}

void plotter(void)
{ float f, x, y;
    int ix, iy, code, count=0, velo;
    char str[50], ch;
    initplotter();
    printf("\nDe schaalfactor f voor een plotter is gewoon-\n");
    printf("lijk erg groot. Als bijvoorbeeld x of y in de\n");
    printf("invoerfile van 0 tot 10 loopt en in plotter-\n");
    printf("coordinaten moet gaan lopen van 0 tot 8000,\n");
    printf("dan zal de schaalfactor f gelijk aan 800 zijn.\n");
    f = scalingfactor();
    printf("\nStandaard snelheid? (J/N): "); ch = getcapital();
    if (ch == 'N')
 { printf("\nGeef snelheid in cm/s, als geheel getal\n");
    printf("(niet kleiner dan 1 of groter dan 127): ");
    scanf("%d", &velo);
    sprintf(str, "VS%d;", velo);
    ser_str(str);
 }

    while (fscanf(fp, "%f %f %d", &x, &y, &code) == 3)
    { ch = getc(fp);
```

```
    if (code < 0 || code > 1 || ch != '\n')
    { printf("Incorrect file-formaat"); exit(1);
    }
    count++;
    ix = (int)(f * x + 0.5);
    iy = (int)(f * y + 0.5);
    if (code != penposition)
    { ser_str(code ? "PD;" : "PU;");
      penposition = code;
    }
    sprintf(str, "PA%04d,%04d;", ix, iy);
    ser_str(str);
  }
  ser_str("PU;");
  printf("\n%d punten\n", count);
}

void initplotter(void)
{ FILE *fplconfg;
  int code, comnr, baud, parity, stopbits, charsize,
     oldvalues=0, retval;
  char answer='Y';
  fplconfg = fopen("plconfig.txt", "r");
  if (fplconfg != NULL)
    oldvalues = (fscanf(fplconfg, "%d %d %d %d %d",
    &comnr, &baud, &parity, &stopbits, &charsize) == 5);
  if (oldvalues)
  { printf("\nParameters voor seriele poort:\n\n");

    printf("Nummer seriele poort: %4d\n", comnr);
    printf("Baud-rate:            %4d\n", baud);
    printf("Pariteit:             %4s\n",
    (parity == 2 ? "even" : parity ? "oneven" : "geen"));
    printf("Aantal stopbits:%4d\n", stopbits);
    printf("Aantal bits per karakter:  %4d bits\n", charsize);
    printf("Is dit correct? (J/N): ");
    answer = getcapital();
    printf("\n");
  }
  if (oldvalues == 0 || answer == 'N')
  { printf(
   "\nNummer van de seriele poort (1 for COM1, 2 for COM2): ");
    scanf("%d", &comnr);
    printf("Baud-rate\n");
```

```
    printf("(110, 150, 300, 600, 1200, 2400, 4800, 9600):");
    scanf("%d", &baud);
    printf("\n\nIndien onbekend, probeer de eerstgenoemde"
    " mogelijkheid:\n");
    printf("Pariteit (0 = geen, 1 = oneven, 2 = even): ");
    scanf("%d", &parity);
    printf("Aantal stopbits? (1 or 2): ");
    scanf("%d", &stopbits);
    printf("Aantal bits per karakter (8 of 7): ");
    scanf("%d", &charsize);
    if (fplconfg != NULL) fclose(fplconfg);
    fplconfg = fopen("plconfig.txt", "w");
    fprintf(fplconfg, "%d %d %d %d %d\n",
         comnr, baud, parity, stopbits, charsize);
    fclose(fplconfg);
  }

  code = (baud == 110 ? 0 :
          baud == 150 ? 1 :
          baud == 300 ? 2 :
          baud == 600 ? 3 :
          baud == 1200 ? 4 :
          baud == 2400 ? 5 :
          baud == 4800 ? 6 :
          baud == 9600 ? 7 : -1);
  if (code == -1) {printf("Baud-rate incorrect"); exit(1);}
  if (parity == 2) parity = 3;
  /* Interne code voor 'even' is 3 */


  port = comnr-1;  /* 0 for com1, 1 for com2 */
  byte = (code << 5) | (parity << 3) |
            ((stopbits-1) << 2) | (charsize - 5);
  retval = bioscom(0, byte, port);
  /* Test op: 'transfer shift register empty',
              'transfer holding register empty',
              'data ready'.
  */
  if ((retval & 0x6100) != 0x6100)
  { printf("Initialisatiefout:\n");
    printf("AH = %02XH\n", retval >> 8);
    exit(1);
  }
  ser_str("IN;PU;"); /* Initialiseer */
  penposition = 0;
}
```

```
float scalingfactor(void)
{ float f;
  printf("\nGeef de schaalfactor f\n");
  printf("(f = 1: op ware grootte\n");
  printf(" f > 1: vergroot\n");
  printf(" f < 1: verkleind): "); scanf("%f", &f);
  getchar();        /* Newline-karakter overgeslagen */
  return f;
}

void ser_str(char *p)
{ while (*p) ser_char(*(p++));
}

void ser_char(char ch)
{ int retval;
  /* Test op: 'Data set ready', 'Clear to send'.
  */
  do
  { retval = bioscom(3, byte, port);
  } while ((retval & 0x0030) != 0x0030);
  bioscom(1, ch, port);
}

char getcapital(void)
{ char ch;
  ch = getche();
  return toupper(ch);
}
```

APPENDIX A: PROGRAMMATEKST VAN MODULE D3D

```
/* D3D: Design in 3 Dimensies.
    Deze module (D3D) moet na compilatie door de linker
    worden samengevoegd met de modulen
    GRPACK (of GRPACK1) (zie paragraaf 4.3),
    HLPFUN (zie paragraaf 4.4)
    TRAFO  (zie paragraaf 3.7)
    Te gebruiken compiler: Turbo C (Borland).
    Geheugenmodel: 'Huge'.
*/
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include <dos.h>
#include "grpack.h"

#define BIG 1e10
#define IBIG 30000
#define EPS 1e-6
#define MAXFACES 2500
#define LIN1 156
#define LIN2 168
#define LIN3 180

void
  initrotate(double a1, double a2, double a3,
          double v1, double v2, double v3, double alpha),
  rotate(double x, double y, double z,
      double *px1, double *py1, double *pz1),
    /* initrotate and rotate are defined in TRAFO */

  init_viewport(void),
  hlpfun(
    float rho, float theta, float phi, float surflimit),
```

```
  coeff(float rho, float theta, float phi),
  ermes(char *s);
  /* init_viewport, hlpfun and coeff zijn gedefinieerd in HLPFUN
  */

int abs(int x);

static void
  aspect(void),
  getstr(int X, int Y, char *str),
  mv(float x, float y, float z),
  dw(float x, float y, float z),
  screencoor(int mode, int i, int *pX, int *pY),
  viewing(float x, float y, float z,
                   double *pxe, double *pye, double *pze),

  display(void),
  plotpoint(int i, int Bold),
  grmes(int meslin, char *s),
  rdfile(int normal),
  wrfile(void),
  cursorcontrol(char direc),
  cursor(float x, float y, float z, int new),
  show(char c),
  faces(void),
  eyepos(void),
  plotsph(float rho, float theta, float phi, int down),
  enquire(int vpos, char *str, float *px),
  textcursor(int col, int line),
  checkfaces(int i),
  infopage(int i),
  helpinfo(void),
  myungetch(char ch),
  line3(int P, int Q),
  checkall(void),
  clear(void),
  zoom(int large),
  reposition(void),
  plotaxes(int checksize),
  fresh(void),
  list_faces(void),
  points(void),
  transform(void),
  check_alloc(int i),
  asksave(void),
  entire(void),
  open_plotfile(void),
  close_plotfile(void),
```

```
  pressanykey(char *str);
static int
  storepoint(int i, float x, float y, float z),
  rdnumber(int *p),
  rdoldnr(int *p),
  getch1(void);
static char
  query(int line, char *s),
  mygetch(void);



struct linsegface
{int i; double a, b, c; struct linsegface *next;};
                                /* Zie module HLPFUN */
struct vertex
{ int inuse;
  float xw, yw, zw;
  double xe, ye, ze;
  struct linsegface *connect; /* Zie module HLPFUN */
} *p;

float x0, y0, z0;

double v11, v12, v13, v21, v22, v23, v32, v33, v43,
  PI, PIdiv18Ø;
float xmin, ymin, xmax, ymax;

static float c1, c2, d, c1first, c2first, dfirst,
  rho, theta, phi, xxx, yyy,
  Xvp_range, Yvp_range, Xvp_center, Yvp_center,
  Xvp_min, Xvp_max, Yvp_min, Yvp_max,
  xwmin, ywmin, zwmin, xwmax, ywmax, zwmax,
  xcur, ycur, zcur, step=Ø.2, zemin, zemax;
static int margin, modified, faces_present,
  zoomin, pridim, bufposition, inside, newcentralpoint,
  numbers=1, numØ, axes, axesØ, auxlines=1, auxØ,
  bold=1, boldØ, plength, psize, faces_entering,
  points_entering, X__max1, progr_arg;
int nmax, **pface, nface, object_present;
static char buffer[5Ø], s2[2], char_avail, filnam[4Ø],
  spaces[6Ø];

main(int argc, char *argv[])
{ int i, lowest, highest;
  long lbytes;
  float threshold, surflimit, xlowest, xhighest;
  char str[3Ø], ch, chØ, ch1;
```

```
if (argc > 1)
{ strcpy(filnam, argv[1]); progr_arg = 1;
}
PI = 4 * atan(1.0);
PIdiv180 = PI/180;
pface = (int **)farcalloc(MAXFACES, sizeof(int *));
psize = sizeof(struct vertex);
lbytes = farcoreleft(); plength = lbytes/(3L * psize);
/* Ongeveer een derde van de vrije geheugenruimte wordt
   gereserveerd voor hoekpunten.
*/
p = (struct vertex *)farcalloc(plength+7, psize);
if (p == NULL) ermes("Geheugenprobleem");
s2[1] = '\0';
margin = 456;
rho = 1000; theta = 20; phi = 75;
coeff(rho, theta, phi);
clearscr();
wrscr(0, 12, "D3D: Design in 3 Dimensies");
wrscr(1, 12, "==========================");
wrscr(2, 12, "Dit programma behoort bij het boek:");
wrscr(4, 12,
"- Ammeraal, L. INTERACTIEVE 3D COMPUTER GRAPHICS,");
wrscr(5, 12, "     Schoonhoven: Academic Service, 1988.");
wrscr(7, 12,
"vertaald uit het Engels; oorspronkelijke titel:");
wrscr(8, 12, "- Interactive 3D Computer Graphics,");
wrscr(9, 12,
"     Chichester: John Wiley & Sons, Chichester, 1988.");
wrscr(11, 12,
"De volgende boeken van dezelfde auteur zijn hiermee verwant:");
wrscr(12, 12, "- De programmeertaal C, Academic Service, 1984.");
wrscr(13, 12,
"- Programming Principles in Computer Graphics, Wiley, 1986.");
wrscr(14, 12, "- Computer Graphics for the IBM PC, Wiley, 1987.");
wrscr(15, 12, "- Programs and Data Structures in C, Wiley, 1987.");

if (progr_arg)
{ wrscr(18, 0,
  "Wenst u verschil in lijndikte? (J/N): ");
  settxtcursor(19, 0); ch = mygetch(); s2[0] = ch;
  if (isalpha(ch)) wrscr(19, 0, s2);
  bold = (ch == 'J' || ch == 'j');
}
wrscr(20, 0,
"Druk op een toets (F1 als hulp gewenst is) ...");
settxtcursor(21, 0);
ch = mygetch();
```

```
if (ch == Ø && getch() == 59 /* F1 */) helpinfo();
initgr(); X__max1 = X__max+1;
for (i=(X__max1 - margin - 16)/8; i>=Ø; i--)
  spaces[i]=' '; /* String van spaties; '\Ø' aan het eind  */
zoom(Ø); /* Stel viewportconstanten in; Ø = klein */
clear(); display();
if (progr_arg) rdfile(Ø);

/* Main loop: */
while
(grmes(LIN1, "Commando:"), textcursor(margin+72, LIN1),
    chØ=mygetch(), ch=toupper(chØ), ch != 'Q')
{ grmes(LIN2, " ");
  grmes(LIN3, " ");
  if (chØ==Ø)
  { ch1=mygetch();
    if (ch1 == 59) /* Functietoets 1 */ helpinfo();
    continue;
  }
  show(ch);
  if (isdigit(ch))
  { if (points_entering) {myungetch(ch); points();} else
    if (faces_entering) {myungetch(ch); faces();} else
    grmes(LIN2, "Begin niet met cijfer");
    continue;
  }

  if (ch == 'I')
  { points_entering = 1; points(); continue;
  }
  points_entering = Ø;

  if (ch == 'F')
  { faces_entering = 1; faces(); modified = 1; continue;
  }
  faces_entering = Ø;

  if (ch=='X' || ch=='Y' || ch=='Z')
  { cursorcontrol(ch); continue;
  }
  if (ch == 'R') rdfile(1); else
  if (ch == 'W') {wrfile(); modified = Ø;} else
  if (ch == 'V') eyepos(); else
  if (ch == 'L') list_faces(); else
  if (ch == 'C')
  { *filnam='\Ø'; asksave(); clear(); display();
  } else
  if (ch == '?')
```

```
{ if (kbhit()) getch();
  grmes(LIN2, "Puntnummer?");
  getstr(margin, LIN3, str);
  if (sscanf(str, "%d", &i) != 1 || i >= nmax ||
                                   p[i].inuse == 0)
  { grmes(LIN2, "Nummer of formaat fout"); continue;
  }
  sprintf(str, "%4.2f %4.2f %4.2f",
               p[i].xw, p[i].yw, p[i].zw);
  grmes(LIN3, str);
} else
if (ch == 'D')
{ if (kbhit()) getch();
  while (nmax > 0 && p[nmax-1].inuse == 0) nmax--;
  xlowest = nmax;
  enquire(LIN2, "Ondergrens: ", &xlowest);
  lowest = (int)(xlowest + 0.1);
  if (lowest < 1) lowest = 1;
  xhighest = xlowest;
  enquire(LIN3, "Bovengrens: ", &xhighest);
  highest = (int)(xhighest + 0.1);
  if (highest >= nmax) highest = nmax - 1;
  if (lowest <= highest)
  { for (i=lowest; i<=highest; i++)
    { p[i].inuse = 0; checkfaces(i);
    }
    checkall(); modified = 1;
    display();
  }
} else
if (ch == 'M')
{ numbers =
  (query(LIN1, "Puntnummers? (J/N):") == 'J');
  axes0 = axes; aux0 = auxlines;
  axes = (query(LIN1, "Assen? (J/N): ") == 'J');
  if (axes)
    auxlines =
    (query(LIN1, "Hulplijnen? (J/N): ") == 'J');
  else auxlines = 0;
  bold = (query(LIN1, "Dik en dun? (J/N):") == 'J');
  if (axes0 && !axes || aux0 && !auxlines) checkall();
  /* Pas beeldgrootte aan, nu assen verwijderd zijn
     en wellicht meer ruimte voor het beeld beschikbaar
     gekomen is.
  */
  if (query(LIN1, "Geheel scherm? (J/N):") == 'J')
    entire(); else
  if (numbers != num0 || axes != axes0 ||
```

```
        auxlines != aux0 || bold != bold0) display();
  } else
  if (ch == 'E') entire(); else
  if (ch == 'P') printgr(0.0, X__max, 0.0, Y__max); else
  if (ch == 'S') enquire(LIN2, "Stapgrootte:", &step); else
  if (ch == 'T') transform(); else
  if (ch == 'H')
  { clearpage();
    to_text();
    if (kbhit()) getch();
    wrscr(0, 0,
    "Wilt u de grensvlakken gekromde oppervlakken"
    " laten benaderen? (J/N): ");
    settxtcursor(0, 68);
    ch = mygetch(); s2[0]=ch;
    if (ch>32) wrscr(0, 68, s2);
    if (ch == 'J' || ch == 'j')
    { wrscr(2, 0, "Geef de 'drempelhoek' in graden. De"
                  " snijlijn van elke twee naburige");
      wrscr(3, 0, "grensvlakken, indien zichtbaar,"
                  " zal alleen getekend worden als de");
      wrscr(4, 0, "hoek tussen de normaalvectoren van"
                  " deze twee vlakken groter is dan de");
      wrscr(5, 0, "gegeven drempelhoek. De waarde"
                  " 35 wordt gebruikt als u alleen op");
      wrscr(6, 0, "de Enter-toets drukt.");
      settxtcursor(6, 22); ch = getchar();
      if (ch == '\n' ||
      (ungetc(ch, stdin), scanf("%f", &threshold) != 1))
         threshold = 35;
      surflimit = cos(threshold*PIdiv180);
    } else surflimit = 1.0;
    aspect();
    initgr();
    hlpfun(rho, theta, phi, surflimit);
    close_plotfile();
    ch = mygetch();
    if (ch == 'P' || ch == 'p')
      printgr(0, X__max, 0, Y__max);
    if (pridim)
    { clearpage(); to_text();
      x_max = 10.0; y_max = 7.0; pridim = 0;
      initgr();
    }
    display();
  } else grmes(LIN2, "Ongeldig commando");
}
asksave();
```

```
  to_text();
}

static void asksave(void)
{ char ch;
  if (modified && nmax + nface > 7)
  { ch = query(LIN2, "Opbergen? (J/N): ");
    if (ch == 'J' || ch == 'j') wrfile();
    grmes(LIN2, " ");
  }
}

static void aspect()
{ char ch;
  wrscr(7, 0, "Wanneer het resultaat op het scherm"
              " zichtbaar is kunt u het met commando P");
  wrscr(8, 0, "op een matrixprinter afdrukken. (Als"
              " u in plaats hiervan op een andere");
  wrscr(9, 0, "toets drukt komt het vorige grafische"
              " scherm weer terug.) Ten aanzien van");
  wrscr(10, 0, "de printer dient u op het volgende"
               " te letten:");
  wrscr(11, 0, "We zeggen dat de 'aspectverhouding'"
               " correct is als een cirkel als een");
  wrscr(12, 0, "echte cirkel wordt afgebeeld en niet"
               " als een ellips. Normaal zal de aspect-");
  wrscr(13, 0, "verhouding correct zijn op het scherm."
               " Als u een correcte aspectverhouding");
  wrscr(14, 0, "op de printer wenst (en een incorrecte"
               " aspectverhouding op het scherm accep-");
  wrscr(15, 0, "teert) druk dan op de toets A. In"
               " plaats hiervan kunt u de letter O typen");
  wrscr(16, 0, "als u een uitvoerfile (xxx.PLT) wenst,"
               " die bijvoorbeeld door programma PLOTHP");
  wrscr(17, 0, "kan worden gelezen om uitvoer op een"
               " HP-plotter te verkrijgen. Als geen van");
  wrscr(18, 0, "beide gewenst wordt, druk dan op een"
               " andere toets ...");
  settxtcursor(19, 0);
  ch = mygetch(); s2[0]=ch;
  if (ch > ' ') wrscr(20, 0, s2);
  if (ch == 'A' || ch == 'a')
  { setprdim(); pridim = 1;
  } else
  if (ch == 'O' || ch =='o') open_plotfile();
}

 static void checkall(void)
```

```
/* Verklein zonodig het begrenzende prisma, nu er een
   punt is verwijderd.
*/
{ int i, count=0;
  float xw, yw, zw, ze;
  xwmin = ywmin = zwmin = zemin = BIG;
  xwmax = ywmax = zwmax = zemax = -BIG;
  for (i=1; i<nmax; i++)
  if (p[i].inuse)
  { count++;
    xw = p[i].xw;
    yw = p[i].yw;
    zw = p[i].zw;
    ze = p[i].ze;
    if (xw < xwmin) xwmin = xw;
    if (xw > xwmax) xwmax = xw;
    if (yw < ywmin) ywmin = yw;
    if (yw > ywmax) ywmax = yw;
    if (zw < zwmin) zwmin = zw;
    if (zw > zwmax) zwmax = zw;
    if (ze < zemin) zemin = ze;
    if (ze > zemax) zemax = ze;
  }
  if (count == 0) clear(); else
  { if (xwmax < 0.5) xwmax = 0.5;
    /* To avoid axes of length 0 */
    if (ywmax < 0.5) ywmax = 0.5;
    if (zwmax < 0.5) zwmax = 0.5;
    x0 = BIG;
    /* Dwingt 'reposition' een nieuw centraal objectpunt
       te berekenen.
    */
    reposition();
  }
}


static void check_alloc(int i)
{ if (i >= plength) ermes("Te hoog puntnummer");
}


static void checkfaces(int i)
{ /* Punt i is zojuist verwijderd. Als er grensvlakken zijn
     waarvan i een hoekpunt was, dan dienen die ook
     verwijderd te worden.
  */
  int j, n, k;
```

```
  if (nface == 0) return;
  grmes(LIN3, "Wacht s.v.p. ...");
  for (j=0; j<nface; j++)
  { if (pface[j] == NULL) continue;
    n = pface[j][0];
    if (n<0) ermes("n<0 in checkfaces");
    for (k=1; k<=n; k++)
    if (abs(pface[j][k]) == i)
    { farfree(pface[j]); pface[j] = NULL; break;
    }
  }
  grmes(LIN3, " ");
}


static void clear(void)
{ static int first=1;
  int i, j;
  axes = auxlines = numbers = 1;
  for (i=0; i<nmax; i++) p[i].inuse = 0;
  p[0].xw = p[0].yw = p[0].zw = 0.0;
  nmax = 1;
  for (j=0; j<nface; j++)
    if (pface[j] != NULL)
    { farfree(pface[j]); pface[j] = NULL;
    }
  nface = 0;
  xwmin = ywmin = zwmin = 0.0;
  xwmax = ywmax = zwmax = 1.0;
  x0 = y0 = z0 = 0.5;
  fresh();
  reposition();
  if (first)
  { c1first = c1; c2first = c2; dfirst = d; first = 0;
  }
  modified = 0; object_present = 0;
}

static void close_plotfile(void)
{ if (fplot != NULL)
  { fclose(fplot);
    fplot = NULL;
  }
}


static void cursor(float xw, float yw, float zw, int new)
{ extern int drawmode;
```

```
  int i, j, dm, cnt;
  static int X, Y, XØ, YØ;
  if (new)
  { inside = 1;
    for (cnt=Ø; cnt<2; cnt++)
    { storepoint(nmax+4, xw, yw, zw);
      storepoint(nmax+5, xw, yw, Ø.Ø);
      screencoor(1, nmax+4, &X, &Y);
      screencoor(1, nmax+5, &XØ, &YØ);
      if (inside) break; else
      { reposition(); display();
      }
    }
    if (cnt == 2) ermes("cnt = 2 in cursor");
    p[nmax+4].inuse = Ø;
    p[nmax+5].inuse = Ø;
  }
  dm = drawmode; drawmode = Ø;
  for (j=-2; j<=2; j+=4)
  for (i=-4; i<=4; i++) dot(X+i, Y+j);
  dot(X-4, Y-1); dot(X-3, Y-1);
  dot(X+3, Y-1); dot(X+4, Y-1);
  dot(X-4, Y); dot(X-3, Y); dot(X+3, Y);
  dot(X+4, Y);  dot(X, Y);
  dot(X-4, Y+1); dot(X-3, Y+1);
  dot(X+3, Y+1); dot(X+4, Y+1);
  draw_line(X, Y, XØ, YØ);
  drawmode = dm;
}


static void cursorcontrol(char direc)
{ char c, cØ, str[3Ø];
  float *axis, xØ, yØ, zØ, dx, dy, dz, dist2, d2;
  int i, j, jumped=Ø;
  if (!auxlines)
  { axes = auxlines = 1; display();
  }
  xcur = ycur = zcur = Ø.Ø;
  cursor(xcur, ycur, zcur, 1);
  axis =
  (direc == 'X' ? &xcur : direc == 'Y' ? &ycur : &zcur);
  grmes(LIN2, "Typ +, -, I, J, of D");
  grmes(LIN3, "(of druk op Enter)");
  while (cØ = mygetch(), c = toupper(cØ),
    c == '+' || c == '-' || c == 'X' || c == 'Y' || c == 'Z'

    || c == 'I' || c == 'J' || c == 'D' || c == Ø)
```

```
{ if (c != 'D') jumped = 0;
  if (c == '+' || c == '-')
  { x0 = xcur; y0 = ycur; z0 = zcur;
    if (c == '-') *axis -= step; else *axis += step;
    cursor(x0, y0, z0, 0);
    cursor(xcur, ycur, zcur, 1);
    sprintf(str, "%5.2f %5.2f %5.2f    ",
                   xcur, ycur, zcur);
    grmes(LIN3, str);
    continue;
  }
  show(c);
  if (c == 'X') {axis = &xcur; direc = c;} else
  if (c == 'Y') {axis = &ycur; direc = c;} else
  if (c == 'Z') {axis = &zcur; direc = c;} else
  if (c == 'I')
  { for (i=1; i<plength; i++) if (p[i].inuse == 0) break;
    check_alloc(i);
    if (i >= nmax) nmax = i+1;
    cursor(xcur, ycur, zcur, 0); /* Verwijder */
    storepoint(i, xcur, ycur, zcur); modified = 1;
    plotpoint(i, 1);
    cursor(xcur, ycur, zcur, 0); /* Teken opnieuw */
  } else
  if (c == 'J')
  { if (nmax == 0)
    { grmes(LIN2, "Geen punten aanwezig");
      cursor(xcur, ycur, zcur, 0);
      grmes(LIN3, " "); return;
    }
    dist2 = BIG;
    for (i=1; i<nmax; i++)
    if (p[i].inuse)
    { dx = p[i].xw - xcur;
      dy = p[i].yw - ycur;
      dz = p[i].zw - zcur;
      d2 = dx*dx + dy*dy + dz*dz;
      if (d2 < dist2) {dist2 = d2; j=i;}
    }
    jumped = 1;
    cursor(xcur, ycur, zcur, 0); /* Verwijder oude cursor */
    xcur = p[j].xw; ycur = p[j].yw; zcur = p[j].zw;
    cursor(xcur, ycur, zcur, 1);
    sprintf(str, "%4d %4.2f %4.2f %4.2f",
                    j, xcur, ycur, zcur);
    grmes(LIN3, str);
  } else
  if (c == 'D')
```

```
  { if (!jumped) grmes(LIN3, "Gebruik eerst J"); else
    { p[j].inuse = 0; cursor(xcur, ycur, zcur, 0);
      checkfaces(j);
      checkall(); modified = 1; jumped = 0;
      display(); cursor(xcur, ycur, zcur, 1);
      grmes(LIN2, "Typ +, -, I, J, ?, D");
      grmes(LIN3, "(of druk op Enter)");
    }
  } else  /* c == 0: Waarschijnlijk pijltjestoets */
  { if (getch() == 59) helpinfo(); else
    { grmes(LIN3, "Verkeerde toets");
    }
  }
  show(direc);
}
cursor(xcur, ycur, zcur, 0); /* Verwijder cursor */
if (c0 != '\n' && c0 != '\r') myungetch(c0);
show(' ');
grmes(LIN2, " ");
grmes(LIN3, " ");



tatic void display(void)
int i, j, k, n, drawn, i_old;
faces_present = 0;
clearpage();
if (axes) plotaxes(1);
if (zoomin) init_viewport();
/* Gedefinieerd in module HLPFUN */
else
{ imove(0, 0); idraw(X__max, 0); idraw(X__max, Y__max);
  idraw(0, Y__max); idraw(0, 0);
  imove(margin-4, 0); idraw(margin-4, Y__max);
  textXY(margin, 5,   "D3D");
  textXY(margin, 20,  "Commando's:");
  textXY(margin, 34,  "X/Y/Z: Cursorbesturing");
  textXY(margin, 46,  "?: Toon coordinaten");
  textXY(margin, 58,  "Mode (Assen etc.)");
  textXY(margin, 70,  "F1 (Help)  | Quit");
  textXY(margin, 82,  "Step       | Clear");
  textXY(margin, 94,  "Insert     | Delete");
  textXY(margin, 106, "Faces      | List");
  textXY(margin, 118, "Read       | Write");
  textXY(margin, 130, "Entire     | Hidden");
  textXY(margin, 142, "Viewpoint  | Transform");
  grmes(LIN3, "Wacht s.v.p. ...");
}
```

```
for (j=0; j<nface; j++)
{ if (pface[j] == NULL) continue;
  faces_present = 1; /* Zie 'list_faces' */
  i_old = abs(pface[j][1]);
  n = pface[j][0];
  if (n<0) ermes("n<0 in display");
  for (k=2; k<=n; k++)
  { i = pface[j][k]; drawn = i >= 0; i = abs(i);
    if (drawn) line3(i_old, i);
    i_old = i;
  }
  i = pface[j][1]; drawn = i >= 0; i = abs(i);
  if (drawn) line3(i_old, i);
}
for (i=1; i<nmax; i++)
if (p[i].inuse)
  plotpoint(i, 0);
if (!zoomin)
{ grmes(LIN1, "Commando:");
  grmes(LIN3, " ");
  num0 = numbers; axes0 = axes; aux0 = auxlines;
  bold0 = bold;
}
if (axes) plotaxes(0);
/* Alleen om letters x, y, z leesbaar te maken */
}

static void dw(float x, float y, float z)
{ int X, Y;
  double xe, ye, ze;
  viewing(x-0.5, y-0.5, z-0.5, &xe, &ye, &ze);
  /* x0 = y0 = z0 = 0.5 */
  X = IX(d*xe/ze+c1);
  Y = IY(d*ye/ze+c2);
  idraw(X, Y);
}


static void enquire(int vpos, char *str, float *px)
{ char snum[40], dum[40];
  int len, slen, j;
  char ch;
  if (kbhit()) getch();
  len = 8*strlen(str);
  if (margin+len > X__max1)
    ermes("String te lang in 'enquire'");
  textXY(margin, vpos, str);
  sprintf(snum, "%3.1f", *px);
```

```
  slen = strlen(snum);
  if (snum[slen-1] == 'Ø' && snum[slen-2] == '.')
    snum[slen-2] = '\Ø';
  if (margin+len+8*strlen(snum) > X__max1)
    ermes("String plus getal te lang in 'enquire'");
  slen = (X__max - margin - len)/8;
  for (j=Ø; j<slen; j++) dum[j] = ' ';

  dum[slen] = '\Ø';
  do
  { j = 1ØØ;
    do textXY(margin+len, vpos, dum);
      while (--j && !kbhit());
    j = 2ØØ;
    do textXY(margin+len, vpos, snum);
      while (--j && !kbhit());
  } while (!kbhit());
  ch = mygetch();
  if (ch == Ø)
  { ch = mygetch();
    if (ch == 59) /* key F1 */
    { helpinfo();
      textXY(margin+len, vpos, dum);
    } else
    if (ch == 3) /* break */
    { to_text(); exit(1);
    }
  } else
  if (ch == '\n' || ch == '\r') return; else myungetch(ch);
  while (1)
  { getstr(margin+len, vpos, snum);
    if (sscanf(snum, "%f", px) == 1) break;
    textXY(margin+len, vpos, dum);
  }
}


static void entire(void)  /* Gebruik het hele scherm */
{ float c1save, c2save, dsave;
  char ch;
  c1save = c1; c2save = c2; dsave = d;
  clearpage();
  to_text();
  if (kbhit()) getch();
  aspect(); /* Kan x_max and y_max wijzigen                */
  initgr(); /* Berekent horfact en vertfact                */
  zoom(1);  /* Wijzigt Xvp_max etc.                        */
  newcentralpoint = Ø; /* Kan in reposition worden veranderd */
```

```
  reposition();  /* Bereken nieuwe c1, c2, d                    */
                 /* Kan ook centraal objectpunt wijzigen        */
  display();     /* Groot beeld                                 */
  close_plotfile();
  ch = mygetch();
  if (ch == 'P' || ch == 'p') printgr(0, X__max, 0, Y__max);

  clearpage(); to_text();
  if (pridim)
  { x_max = 10.0; y_max = 7.0; pridim = 0;
  }
  initgr();
  zoom(0);
  if (newcentralpoint) reposition();
           /* Bereken nieuwe c1, c2 en d   */
  else
  { c1 = c1save; c2 = c2save; d = dsave;
  }
  display();
}


void ermes(char *s)
{ char ch;
  if (in_textmode) {printf(s); exit(1);}
  /* 'in_textmode' is an externe variabele */
  to_text();
  printf(s);
  if (!modified || nmax + nface <= 7) exit(1);
  printf("\n\nD3D zal stoppen.\n");
  printf("Wilt u het voorwerp opbergen? (J/N): ");
  ch = getche();
  if (ch == 'J' || ch == 'j')
  { initgr(); wrfile(); to_text();
  }
  exit(1);
}


static void eyepos(void)
{ float rho_1, theta_1, phi_1, rho1, theta1, phi1,
    dtheta, dphi, rho2;
  int i, n = 5;
  rho_1 = rho; theta_1 = theta; phi_1 = phi;
  rho = 1000; theta = 20; phi = 75;
  coeff(rho, theta, phi);
  clearpage();
  c1 = c1first; c2 = c2first; d = dfirst;
```

```
  mv(0.0, 0.0, 0.0); dw(1.0, 0.0, 0.0); text("x");
  mv(0.0, 0.0, 0.0); dw(0.0, 1.0, 0.0); text("y");
  mv(0.0, 0.0, 0.0); dw(0.0, 0.0, 1.0); text("z");
  mv(0.0, 0.0, 0.0);
  rho1 = 1.0; theta1 = 40; phi1 = 40;
  rho2 = rho1*sin(theta1*PIdiv180);
  plotsph(rho2, theta1, 90.0, 1);
  plotsph(rho1, theta1, phi1, 1);
  dw(0.0, 0.0, 0.0);
  rho1 = 0.9;
  dtheta = theta1/n;
  rho2 = 0.9 * rho2;
  mv(rho2, 0.0, 0.0);
  for (i=1; i<=n; i++) plotsph(rho2, i*dtheta, 90.0, 1);
  dphi = phi1/n;
  plotsph(rho1, 0.0, 0.0, 0);
  for (i=1; i<=n; i++) plotsph(rho1, theta1, i*dphi, 1);
  plotsph(0.5*rho1, theta1, phi1, 0); text("rho");
  plotsph(rho2, 0.4*theta1, 90.0, 0); text("theta");
  plotsph(rho1, theta1, 0.4*phi1, 0); text("phi");
  plotsph(1.2*rho1, theta1, phi1, 0); text("Oog");
  textXY(margin, 15, "Geef rho, theta, phi");
  textXY(margin, 30, "(theta en rho in gr.)");
  textXY(margin, 45, "Als u op Enter drukt");
  textXY(margin, 60, "worden de getoonde");
  textXY(margin, 75, "waarden gebruikt.");
  enquire(90,  "rho   = ", &rho_1);
  enquire(105, "theta = ", &theta_1);
  enquire(120, "phi   = ", &phi_1);
  rho = rho_1; theta = theta_1; phi = phi_1;
  coeff(rho, theta, phi);
  x0 = BIG;
  /* Dwingt 'reposition' een nieuw centraal objectpunt
     te berekenen.
  */
  reposition(); display();
}


static void faces(void)
{ int i, n=0, draw, i_old=0;
  char str[30];
  double xA, yA, zA, xB, yB, zB, xC, yC, zC, a, b, c, h;
  if ((pface[nface] = (int *)farcalloc(5, sizeof(int)))
    == NULL)
    ermes("Niet genoeg geheugen");
  grmes(LIN2, "Nrs., daarna een punt:");
  textcursor(margin, LIN3);
```

```
bufposition = -1;
while (1)
{ if (rdnumber(&i) == 0) break;
  draw = i >= 0; i = abs(i);
  n++;
  if (p[i].inuse == 0)
  { sprintf(str, "Ongedef. punt: %6d", i);
    pressanykey(str); display();
    faces_entering = 0; return;
  }
  if (n == 1)
  { xA = p[i].xw; yA = p[i].yw; zA = p[i].zw;
  } else
  if (n == 2)
  { xB = p[i].xw; yB = p[i].yw; zB = p[i].zw;
  } else
  if (n == 3)
  { xC = p[i].xw; yC = p[i].yw; zC = p[i].zw;
    h = xA * (yB*zC - yC*zB) -
      xB * (yA*zC - yC*zA) +
      xC * (yA*zB - yB*zA);
    a = yA * (zB-zC) - yB * (zA-zC) + yC * (zA-zB);
    b = -(xA * (zB-zC) - xB * (zA-zC) + xC * (zA-zB));
    c = xA * (yB-yC) - xB * (yA-yC) + xC * (yA-yB);
  } else
  if (fabs(a*p[i].xw + b*p[i].yw + c*p[i].zw - h) >
                              0.001 * fabs(h) + EPS)
  { pressanykey("Niet in hetzelfde vlak");
    display(); /* Verwijder onjuiste lijnen */
    faces_entering = 0; return;
  }
  if (n > 1 && draw) line3(i_old, i);
  i_old = i;
  if (n > 4)
  { if ((pface[nface] = (int *)farrealloc(
           pface[nface], (n+1) * sizeof(int))) == NULL)
    ermes("Niet genoeg geheugen");
  }
  pface[nface][n] = (draw ? i : -i);
}
if (n == 0)
{ grmes(LIN2, "Ongeldig nummer");
  farfree(pface[nface]); faces_entering = 0; return;
}
i = pface[nface][1];
if (i >= 0)
line3(i_old, i);
pface[nface][0] = n; nface++;
```

```
  if (nface == MAXFACES)
  { grmes(LIN2, "Te veel grensvlakken"); nface--;
  }
}


static void fresh(void)
{ int i, X, Y;
  xmin = ymin = zemin = BIG;
  xmax = ymax = zemax = -BIG; /* wordt bijgewerkt */
  /* Compute xmin, xmax, ymin, ymax:  */
  storepoint(nmax, 0.0, 0.0, 0.0);
  storepoint(nmax+1, xwmax, 0.0, 0.0);
  storepoint(nmax+2, 0.0, ywmax, 0.0);
  storepoint(nmax+3, 0.0, 0.0, zwmax);
  screencoor(0, nmax, &X, &Y);
  screencoor(0, nmax+1, &X, &Y);
  screencoor(0, nmax+2, &X, &Y);
  screencoor(0, nmax+3, &X, &Y);
  for (i=nmax; i<nmax+4; i++) p[i].inuse = 0;
}


static int getch1(void)  /* Alleen in 'rdnumber' gebruikt */
{ char ch;
  if (bufposition < 0)
  { getstr(margin, LIN3, buffer); bufposition = 0;
  }
  ch = buffer[bufposition++];
  if (8*bufposition >= X__max1 - margin)
  { getstr(margin, LIN3, buffer);
    ch = buffer[0]; bufposition = 1;
  } else
  if (ch == '\0')
  { bufposition = -1; ch = '\n';
  }
  return ch;
}


static void getstr(int X, int Y, char *str)
{ char ch;
  int i=0, first=1, j, k;
  while (1)
  { j=X+8*i;
    if (j >= X__max1-8) break;
    if (first)
```

```
    { for (k=X; k<X__max1-8; k+=8) textXY(k, Y, " ");
      first=0;
    }
    textcursor(j, Y);
    ch=mygetch();
    if (ch == '\n' || ch == '\r') break;
    if (ch==8) /* backspace */
    { if (--i<0) i=0;
    } else
    { str[i] = s2[0] = ch;
      textXY(j, Y, s2);
      i++;
    }
  }
  str[i] = '\0';
}


static void grmes(int meslin, char *s)
{ int len;
  len = 8*strlen(s);
  if (!in_textmode) /* external variable */
  { textXY(margin, meslin, spaces);
    if (margin+len > X__max1)
    { to_text(); printf("In grmes:\n");
      printf("s='%s' len=%d margin=%d", s, len, margin);
      exit(1);
    }
    textXY(margin, meslin, s);
  } else
  printf("%s\n", s);
}


static void helpinfo(void)
{ int i=1, was_in_grmode;
  was_in_grmode = !in_textmode; /* externe variabele */
  settxtcursor(24, 44);
  if (was_in_grmode)  { clearpage(); to_text();}
  do
  { infopage(i);
    if (i == 1)
    wrscr(24, 0, "Druk op toets om verder te gaan ..."); else
    { wrscr(23, 0,
      "Druk op 'pijltje omhoog' voor de vorige informatiepagina");
      wrscr(24, 0, "of op een andere toets om verder te gaan ...");
    }
    if (mygetch() == 0 && mygetch() == 72)
```

```
    { if (--i == 0) i = 1;
    } else i++;
  } while (i < 4);
  if (was_in_grmode)
  { initgr(); display();
  }
}



static void infopage(int i)
{ if (i == 1)
  {
  clearscr();
  wrscr(0, 67, "Info-pagina 1");
  wrscr(1, 67, "(3 pagina's)");
  wrscr(2, 0,  "D3D: Design in 3 Dimensies,");
  wrscr(3, 0, "Geprogrammeerd door: L. Ammeraal");
  wrscr(5, 0, "Programma D3D stelt u in staat realisti"
               "sche afbeeldingen van 3D voorwerpen te");
  wrscr(6, 0, "maken. Het gemakkelijkst gaat dit met"
               " een invoerfile gemaakt met dit program-");
  wrscr(7, 0, "ma of op een andere manier. In plaats"
               " hiervan kunt u ook zonder invoerfile");
  wrscr(8, 0, "beginnen en zelf punten invoeren met"
               " commando I, hetzij direct, hetzij met");
  wrscr(9, 0, "cursorbesturing. In eerstgenoemd geval"
               " wordt commando I gevolgd door een");
  wrscr(10, 0, "positief geheel getal n en de 3D coordi"
               "naten x, y, z van een nieuw punt. In het");
  wrscr(11, 0, "tweede geval wordt bij het definieren van"
               " een punt een cursor gebruikt. U drukt");
  wrscr(12, 0, "dan op een van de toetsen X, Y of Z,"
               " gevolgd door + of - om de cursor te be-");
  wrscr(13, 0, "wegen. Bij cursorbesturing voegt I een"
               " nieuw punt toe en laat J de cursor naar");
  wrscr(14, 0, "het dichtstbijzijnde bestaande punt"
               " springen. U kunt met commando S de stap-");
  wrscr(15, 0, "grootte voor cursorverplaatsingen veran"
               "deren. Om de coordinaten van een punt te");
  wrscr(16, 0, "tonen typt u een vraagteken gevolgd door"
               " het puntnummer. Met commando D kunt u");
  wrscr(17, 0, "alle punten met nummers in een gegeven"
               " bereik verwijderen. Ook bij cursorbe-");
  wrscr(18, 0, "sturing werkt D, maar dan wordt alleen"
               " een enkel punt verwijderd, dat met J");
  wrscr(19, 0, "wordt geselecteerd. Het mode-commando M"
               " dient om puntnummers, assen, verticale");
```

```
wrscr(20, 0, "projectielijnen te tonen of weg te laten"
             " en om iets over lijndikte en scherm-");
wrscr(21, 0, "grootte op te geven.");
} else
if (i == 2)
{
clearscr();
wrscr(0, 67, "Info-pagina 2");
wrscr(1, 68, "(3 pagina's)");
wrscr(2, 0, "Nadat u enige punten hebt ingevoerd, kunt"
            " u grensvlakken definieren en lijnen");
wrscr(3, 0, "tekenen door middel van commando F, gevo"
            "lgd door puntnummers en dan gevolgd door");
wrscr(4, 0, "een punt (.), zoals bijvoorbeeld in:");
wrscr(5, 0, "   F");
wrscr(6, 0, "   1 2 3 4 5.");
wrscr(7, 0, "Als F gevolgd wordt door meer dan twee"
            " puntnummers, dan moeten deze de hoekpun-");
wrscr(8, 0, "ten van een veelhoek aanduiden, die alle"
            " in hetzelfde vlak liggen, zie ook com-");
wrscr(9, 0, "mando R. Na commando H, zie hieronder,"
            " fungeren deze veelhoeken als de grens-");
wrscr(10, 0, "vlakken van een driedimensionaal voorwerp;"
             " de hoekpunten moeten (bekeken van");
wrscr(11, 0, "buiten het voorwerp) anti-kloksgewijs"
             " worden opgegeven. Een vergissing met");
wrscr(12, 0, "commando F kan worden gecorrigeerd met"
             " commando L. Commando R leest een voor-");
wrscr(13, 0, "werp van een file en commando W schrijft"
             " het naar een file. Om het oogpunt te");
wrscr(14, 0, "veranderen gebruikt u commando V. Alle"
             " verborgen lijnen worden verwijderd door");
wrscr(15, 0, "commando H. Door de gevraagde 'drempelhoek'"
             " groter dan 0 te kiezen, kunt u aan-");
wrscr(16, 0, "grenzende vlakken (met een ingesloten ho"
             "ek van bijna 180 gr.) een gekromd opper-");
wrscr(17, 0, "vlak laten benaderen. De snijlijn van"
             " twee zulke grensvlakken wordt alleen ge-");
wrscr(18, 0, "tekend als de hoek tussen hun normaalvec"
             "toren groter is dan de gegeven 'drem-");
wrscr(19, 0, "pelhoek'. Commando E maakt dat het"
             " hele scherm voor het beeld wordt gebruikt.");
wrscr(20, 0, "De commando's E en H kunnen ook uitvoer op"
             " de printer en de plotter leveren.");
} else
if (i == 3)
{
clearscr();
```

```
  wrscr(0, 67, "Info-pagina 3");
  wrscr(1, 68, "(3 pagina's)");
  wrscr(2, 0, "Er zijn enkele voorbeeldfiles, namelijk"
              " EXAMPLE1.DAT enz., die de structuur van");
  wrscr(3, 0, "'objectfiles' tonen, zoals deze door de "
              "commando's W en R respectievelijk worden");
  wrscr(4, 0, "geschreven en gelezen. Eerst worden alle"
              " punten opgegeven in de vorm n x y z,");
  wrscr(5, 0, "waarin n een puntnummer is. Dan kan het"
              " woord 'Faces:' volgen, met daarna rijen");
  wrscr(6, 0, "puntnummers, zie ook commando F. Iedere"
              " rij wordt gevolgd door een punt (.) of");
  wrscr(7, 0, "door het karakter #. De punten in elke"
              " rij zijn de hoekpunten van een veelhoek,");
  wrscr(8, 0, "dat, na commando H, fungeert als grens"
              "vlak. De zijden van zulke veelhoeken,");
  wrscr(9, 0, "indien zichtbaar, zijn te trekken lijn"
              "stukken, tenzij er negatieve puntnummers");
  wrscr(10, 0, "zijn. In bijvoorbeeld '8 3 -5 2 7. wordt"
               " zijde 3 5 niet getekend, ook al is hij");
  wrscr(11, 0, "zichtbaar. Een rij van maar twee punten"
               " stelt een los lijnstuk voor. Commando T");
  wrscr(12, 0, "biedt vier typen transformaties, name"
               "lijk rotatie, translatie, schaling en");
  wrscr(13, 0, "spiegeling.");
  wrscr(15, 0, "De eenvoudigste manier om te beginnen"
               " is met commando R een gegeven file, zoals");
  wrscr(16, 0, "EXAMPLE1.DAT, in te lezen; u kunt dan met"
               " commando V het oogpunt veranderen en");
  wrscr(17, 0, "met commando E overgaan op het hele sche"
               "rm. Daarna kan commando H, voor het weg-");
  wrscr(18, 0, "laten van verborgen lijnen, worden aanbe"
               "volen. Als u vervolgens zelf een teke-");
  wrscr(19, 0, "ning wilt gaan maken (met X, Y, Z, F, et"
               "c.), dient u eerst met commando C het");
  wrscr(20, 0, "scherm schoon te maken.");
  }
}


static void line3(int P, int Q)
{ float ze1, ze2, zrange=zemax-zemin+1e-15;
  int X1, Y1, X2, Y2, i1, i2, j1, j2;
  char str[10];
  static int dx[]={0, 1, -1, 0, 0},
             dy[]={0, 0, 0, 1, -1},
             n[]={5, 4, 3, 2, 1};
  screencoor(1, P, &X1, &Y1); /* Berekent xxx en yyy */
```

```
  if (zoom && fplot != NULL) move(xxx, yyy);
  screencoor(1, Q, &X2, &Y2);
  if (zoom && fplot != NULL) draw(xxx, yyy);
  if (!zoom || bold)
  { ze1 = p[P].ze; ze2 = p[Q].ze;
    i1 = (int)((ze1-zemin)/zrange * 4.999);
    i2 = (int)((ze2-zemin)/zrange * 4.999);
    if (i1<0 || i2<0 || i1>4 || i2>4)
    { to_text(); printf("i1=%d i2=%d\n", i1, i2);
      printf("P = %d\n", P);
      printf("Q = %d\n", Q);
      printf("ze1=%f ze2=%f zemin=%f zrange=%f\n",
              ze1, ze2, zemin, zrange);
      exit(1);
    }
    for (j1=0; j1<n[i1]; j1++)
    for (j2=0; j2<n[i2]; j2++)
    draw_line(X1+dx[j1], Y1+dy[j1], X2+dx[j2], Y2+dy[j2]);
  } else draw_line(X1, Y1, X2, Y2);
  if (numbers)
  { sprintf(str, "%d", P); imove(X1-2, Y1-5); text(str);
    sprintf(str, "%d", Q); imove(X2-2, Y2-5); text(str);
  }
}


static void list_faces(void)
{ int j, i, nch, nch0, count=0, n, nchmax;
  char str[50], ch;
  nchmax = (X__max-margin)/8;
  for (j=0; j<nface; j++)
  { if (pface[j] == NULL) continue;
    count++;
    n = pface[j][0]; nch = 0;
    if (n<0) ermes("n<0 in list_faces");
    for (i=1; i<=n; i++)
    { nch0=nch;
      nch += sprintf(str+nch, " %d", pface[j][i]);
      if (nch > nchmax) { str[nch0] = '\0'; break; }
    }
    s2[0] = '.';
    grmes(LIN2, str);
    textXY(margin + 8*strlen(str), LIN2, s2);
    ch = query(LIN3, "OK? (J/N): ");
    if (ch == 'N')
    { farfree(pface[j]); pface[j] = NULL; display();
      if (!faces_present &&
          (! axes || !auxlines || !numbers))
```

```
      { axes = auxlines = numbers = 1; display();
      }
    } else
    if (ch != 'J') break;
  }
  if (count == 0) grmes(LIN2, "Geen grensvlakken");
             else grmes(LIN2, " ");
  grmes(LIN3, " ");
}


int matherr(struct exception *a)
{ if (a->type == DOMAIN)
  { if (strcmp(a->name, "sqrt") == 0)
      ermes("Wortel uit negatief getal");
  }
  ermes("Floating-point: fout");
  return(0); /* Wordt niet uitgevoerd i.v.m. ermes */
}


static void mv(float x, float y, float z)
{ int X, Y;
  double xe, ye, ze;
  viewing(x-0.5, y-0.5, z-0.5, &xe, &ye, &ze);
  /* x0 = y0 = z0 = 0.5 */
  X = IX(d*xe/ze+c1);
  Y = IY(d*ye/ze+c2);
  imove(X, Y);
}


static char mygetch(void)
{ char ch;
  ch = getch();
  if (ch == 3)   /* Ctrl-C  of Ctrl-Break */
  { if (!in_textmode) to_text();
    exit(1);
  }
  char_avail = 0;
  return ch;
}

static void myungetch(char ch)
{ ungetch(ch); char_avail = 1;
}
```

```
static void open_plotfile(void)
{ char *p, *q, plfilnam[40];
  if (*filnam)
  { p=filnam; q=plfilnam;
    while (*p && *p != '.') *q++ = *p++;
    strcpy(q, ".plt");
  } else strcpy(plfilnam, "noname.plt");
  fplot = fopen(plfilnam, "w");
  if (fplot == NULL) ermes("Kan plotfile niet openen");
}


static void plotaxes(int checksize)
{ int X0, Y0, X1, Y1, X2, Y2, X3, Y3, cnt, i;
  if (checksize)
  { storepoint(nmax, 0.0, 0.0, 0.0);
    storepoint(nmax+1, xwmax, 0.0, 0.0);
    storepoint(nmax+2, 0.0, ywmax, 0.0);
    storepoint(nmax+3, 0.0, 0.0, zwmax);
    screencoor(0, nmax, &X0, &Y0);
    screencoor(0, nmax+1, &X1, &Y1);
    screencoor(0, nmax+2, &X2, &Y2);
    screencoor(0, nmax+3, &X3, &Y3);
    reposition();
    inside = 1;
  }
  storepoint(nmax, 0.0, 0.0, 0.0);
  storepoint(nmax+1, xwmax, 0.0, 0.0);
  storepoint(nmax+2, 0.0, ywmax, 0.0);
  storepoint(nmax+3, 0.0, 0.0, zwmax);
  screencoor(1, nmax, &X0, &Y0);
  screencoor(1, nmax+1, &X1, &Y1);
  screencoor(1, nmax+2, &X2, &Y2);
  screencoor(1, nmax+3, &X3, &Y3);
  for (i=nmax; i<nmax+4; i++) p[i].inuse = 0;
  if (checksize && !inside)
  { to_text();
    printf("not inside in plotaxes\n");
    printf("d=%f c1=%f c2=%f\n", d, c1, c2);
    printf("Xvp_min=%f Xvp_max=%f  Yvp_min=%f Yvp_max=%f\n",

           Xvp_min, Xvp_max, Yvp_min, Yvp_max);
    for (cnt=nmax; cnt<nmax+4; cnt++)
    { printf("cnt=%d xe=%f ye=%f ze=%f X=%d Y=%d\n",
             cnt, p[cnt].xe, p[cnt].ye, p[cnt].ze,
             IX(d*p[cnt].xe/p[cnt].ze+c1),
             IY(d*p[cnt].ye/p[cnt].ze+c2));
    }
```

```
    exit(0);
  }
  imove(X0, Y0); idraw(X1, Y1); text("x");
  imove(X0, Y0); idraw(X2, Y2); text("y");
  imove(X0, Y0); idraw(X3, Y3); text("z");
}


static void plotpoint(int i, int Bold)
{ char str[30];
  int X, Y, Xxy0, Yxy0, cnt;
  inside = 1;
  for (cnt=0; cnt<2; cnt++)
  { screencoor(1, i, &X, &Y);
    if (auxlines)
    { storepoint(nmax+6, p[i].xw, p[i].yw, 0.0);
      screencoor(1, nmax+6, &Xxy0, &Yxy0);
      p[nmax+6].inuse = 0;
    }
    if (inside) break; else
    { reposition(); display();
    }
  }
  if (cnt == 2) ermes("cnt = 2 in plotpoint");
  if (auxlines) draw_line(Xxy0, Yxy0, X, Y);
  dot(X, Y);
  if (numbers)
  { sprintf(str, "%d", i); imove(X-2, Y-5); text(str);
  } else
  if (nface == 0 || Bold)
  { dot(X-1, Y-1); dot(X, Y-1); dot(X+1, Y-1);
    dot(X-1, Y);                dot(X+1, Y);
    dot(X-1, Y+1); dot(X, Y+1); dot(X+1, Y+1);
  }
}


static void
plotsph(float rho, float theta, float phi, int down)
{ float x, y, z, rcosphi, rsinphi,
    costh, sinth;
  theta = theta * PIdiv180; phi = phi * PIdiv180;
  rcosphi = rho * cos(phi); rsinphi = rho * sin(phi);
  costh = cos(theta); sinth = sin(theta);
  x = rsinphi * costh; y = rsinphi * sinth; z = rcosphi;
  if (down) dw(x, y, z); else mv(x, y, z);
}
```

```
static void points(void)
{ char str[50];
  int i;
  float x, y, z;
  grmes(LIN2, "n  x  y  z:");
  getstr(margin, LIN3, str);
  if
  (sscanf(str, "%d %f %f %f", &i, &x, &y, &z) != 4 ||
    i <= 0)
  { grmes(LIN2, "Ongeldig getalformaat");
    points_entering = 0;
    return;
  }
  check_alloc(i);
  if (i >= nmax) nmax = i+1;
  if (storepoint(i, x, y, z)) {checkall(); display();}
  plotpoint(i, 1); modified = 1;
}

static void pressanykey(char *str)
{ grmes(LIN2, str);
  grmes(LIN3, "Druk op een toets ...");
  mygetch();
}

static char query(int line, char *s)
{ int pos;
  char ch;
  grmes(line, s);
  pos = margin + 8 * strlen(s);
  textcursor(pos, line); ch = mygetch();
  s2[0] = ch; ch = toupper(ch);
  textXY(pos, line, s2);
  return ch;
}


static void rdfile(int normal)
{ FILE *fp;
  int ch, i_old=0, base, lowlimit,
     i, n, drawn, X, Y;
  float x, y, z, i_float;
  char str[40];
  modified = 0; /* File en datastructuur identiek! */
  if (normal)
  { if (object_present)
    { ch = query(LIN2, "Vervangen? (J/N): ");
      if (ch == 'J' || ch == 'j') clear();
```

```
      else modified = 1;
    }
    if (kbhit()) getch();
    grmes(LIN2, "Invoerfile: ");
    getstr(margin, LIN3, filnam);
  }
  fp = fopen(filnam, "r");
  if (fp == NULL) {grmes(LIN2, "Kan file niet openen"); return;}
  grmes(LIN3, "Wacht s.v.p. ...");

  base = nmax - 1; inside = 1; lowlimit = 32767;
  while
  (fscanf(fp, "%f %f %f %f", &i_float, &x, &y, &z) == 4)
  { if (getc(fp) != '\n')
    { grmes(LIN2, "Onjuist fileformaat");
      grmes(LIN3, "Gebruik: n x y z");
      fclose(fp); return;
    }
    i = (int)(i_float + 0.001);
    if (i <= 0)
    { grmes(LIN3, "Puntnr. niet positief");
      fclose(fp); return;
    }
    i += base;
    if (i < lowlimit) lowlimit = i;
    check_alloc(i);
    if (i >= nmax) nmax = i+1;
    storepoint(i, x, y, z);
    screencoor(0, i, &X, &Y);
  }
  if (base && lowlimit < 32767)
  { grmes(LIN1, "Nieuwe punten:");
    sprintf(str, "%d-%d", lowlimit, nmax - 1);
    pressanykey(str);
  }
  inside = 1;
  for (i=base; i<nmax; i++)
    if (p[i].inuse) screencoor(1, i, &X, &Y);

  if (!inside) reposition();
  do ch = getc(fp); while (isspace(ch));
  axes = numbers = (ch != 'F' && ch != 'f');
  if (!axes) auxlines = 0;
  display();
  if (!axes) /* Dat wil zeggen als er grensvlakken zijn */
  { do ch = getc(fp); while (ch != '\n' && ch != EOF);
    /* Sla invoerregel met keyword "Faces" over */
    while (fscanf(fp, "%d", &i) == 1)
```

```
    { if ((pface[nface] = (int *)farcalloc(5, sizeof(int)))
        == NULL)
        ermes("Niet genoeg geheugen");
      n = 0;
      while (1)
      { if (n > 0) {if (fscanf(fp, "%d", &i) <= 0) break;}
        n++;
        drawn = i >= 0; i = abs(i) + base;
        if (p[i].inuse == 0)
        { sprintf(str, "Ongedef. punt: %6d", i);
          grmes(LIN3, str); fclose(fp); return;
        }
        if (n > 1 && drawn) line3(i_old, i);
        i_old = i;
        if (n > 4)
        { if ((pface[nface] = (int *)
          farrealloc(pface[nface], (n+1) * sizeof(int)))
          == NULL)
          ermes("Niet genoeg geheugen");
        }
        pface[nface][n] = (drawn ? i : -i);
      }
      ch = getc(fp);
      if (ch != '#' && ch != '.')
      { grmes(LIN3, "Punt (.) of # verwacht"); fclose(fp);
        return;
      }
      if (n > 2)
      { i = pface[nface][1];
        if (i >= 0) line3(i_old, i);
      }
      pface[nface][0] = n; nface++;
      if (nface == MAXFACES) ermes("Te veel grensvlakken");
    }
    ch = getc(fp);
  } else numbers = 1;
  if (ch != EOF) grmes(LIN2, "Incorrect file-formaat"); else
     grmes(LIN2, " ");
  grmes(LIN3, " ");
  fclose(fp);
}


static int rdnumber(int *p)
/* Alleen in 'faces' en in 'transform' aangeroepen */
{ int i, neg, d, i0;
  char ch;
  *p = 0;
```

```
  do ch = getch1(); while (isspace(ch));
  neg = ch == '-';
  if (neg || ch == '+') ch = getch1();
  if (!isdigit(ch)) return 0;
  i = ch - '0';
  while (ch = getch1(), isdigit(ch))
  { d = ch - '0';
    i0 = i; i = 10 * i + d;
    if (i < i0) {grmes(LIN3, "Te veel cijfers"); return 0;}
  }
  *p = (neg ? -i : i);
  if (bufposition > 0) bufposition--;
  /* i.e. myungetch1(ch) */
  return 1;
}


static int rdoldnr(int *q)
{ int code, i, pnr;
  code = rdnumber(&i); pnr = abs(i); *q = i;
  if
  (code == 0 || pnr >= nmax ||   pnr  && p[pnr].inuse == 0)
  { grmes(LIN2, "Incorrect puntnr."); mygetch(); return 0;

  } else return 1;
}


static void reposition(void)
/* Vereist dat xwmin etc. correct zijn */
{ float Xrange, Yrange, fx, fy, Xcenter, Ycenter,
    x1, y1, z1, q=0.4, xtol, ytol, ztol;
  int i, X, Y;
  x1 = 0.5 * (xwmin + xwmax);
  y1 = 0.5 * (ywmin + ywmax);
  z1 = 0.5 * (zwmin + zwmax);
  xtol = q * (xwmax - xwmin);
  ytol = q * (ywmax - ywmin);
  ztol = q * (zwmax - zwmin);
  if (fabs(x0-x1) > xtol ||
      fabs(y0-y1) > ytol ||
      fabs(z0-z1) > ztol)
  { grmes(LIN3, "Wacht s.v.p. ...");
    x0 = x1; y0 = y1; z0 = z1;
    newcentralpoint = 1; /* Zie commando 'E' */
    fresh();
    for (i=0; i<nmax+6; i++)  /* Zie 'clear' en 'cursor' */
```

```
    { if (p[i].inuse)
      { storepoint(i, p[i].xw, p[i].yw, p[i].zw);
        screencoor(0, i, &X, &Y);
        /* Compute new xmin etc. */
        if (axes)
        { storepoint(nmax+6, p[i].xw, 0.0, 0.0);
          screencoor(0, nmax+6, &X, &Y);
          storepoint(nmax+6, 0.0, p[i].yw, 0.0);
          screencoor(0, nmax+6, &X, &Y);
          storepoint(nmax+6, 0.0, 0.0, p[i].zw);
          screencoor(0, nmax+6, &X, &Y);
        }
        if (auxlines)
        { storepoint(nmax+6, p[i].xw, p[i].yw, 0.0);
          screencoor(0, nmax+6, &X, &Y);
        }
      }
    }
    p[nmax+6].inuse = 0;
    grmes(LIN3, " ");
  }
  Xrange = xmax - xmin; Yrange = ymax - ymin;
  if (Xrange < 1e-12) Xrange = 1e-12;
  if (Yrange < 1e-12) Yrange = 1e-12;
  Xcenter = 0.5*(xmin+xmax); Ycenter = 0.5*(ymin+ymax);
  fx = Xvp_range/Xrange; fy = Yvp_range/Yrange;
  d = (fx < fy ? fx : fy);
  if (!zoomin) d *= 0.85;
  /* Maak ook wat ruimte voor nieuwe punten */
  c1 = Xvp_center - d*Xcenter; c2 = Yvp_center - d*Ycenter;
}

static void
screencoor(int mode, int i, int *pX, int *pY)
{ double xe, ye, ze, xs, ys;
  xe = p[i].xe; ye = p[i].ye; ze = p[i].ze;
  xs = xe/ze;  ys = ye/ze;
  if (xs < xmin) xmin = xs;
  if (xs > xmax) xmax = xs;
  if (ys < ymin) ymin = ys;
  if (ys > ymax) ymax = ys;
  if (mode == 0) return;
  xxx = d*xs+c1; yyy = d*ys+c2;
  if (!zoomin)
  { if (xxx < Xvp_min || xxx > Xvp_max ||
        yyy < Yvp_min || yyy > Yvp_max) inside = 0;
  }
  *pX = IX(xxx); *pY = IY(yyy);
```

```
}


static void show(char c)
{ if (!zoomin)
  { s2[0] = c; textXY(margin + 72, LIN1, s2);
  }
}


static int storepoint(int i, float xw, float yw, float zw)
{ int oldpoint=0;
  double xe, ye, ze;
  if (i < nmax && p[i].inuse)  oldpoint = 1;
  p[i].xw = xw; p[i].yw = yw; p[i].zw = zw;
  if (p[i].inuse == 0) p[i].inuse = 1;
  viewing(xw-x0, yw-y0, zw-z0, &xe, &ye, &ze);
  p[i].xe = xe; p[i].ye = ye; p[i].ze = ze;
  if (xw < xwmin) xwmin = xw;
  if (xw > xwmax) xwmax = xw;
  if (yw < ywmin) ywmin = yw;
  if (yw > ywmax) ywmax = yw;
  if (zw < zwmin) zwmin = zw;
  if (zw > zwmax) zwmax = zw;
  if (ze < zemin) zemin = ze;
  if (ze > zemax) zemax = ze;
  if (i < nmax) object_present = 1;
  return oldpoint;
  /* 'storepoint' geeft 1 terug als punt i al bestaat;
     normaal is dat niet het geval en wordt 0 teruggegeven.
  */
}


static void textcursor(int col, int line)
{ static char underline[2]="_", dum[2]=" ";
  int j;
  if (char_avail) return;
  do
  { j = 100;
    do textXY(col, line, underline);
      while (--j && !kbhit());
    j = 200;
    do textXY(col, line, dum); while (--j && !kbhit());
  } while (!kbhit());
}
```

```
static void transform(void)
{ char ch, str[30];
  int P=-1, Q=-1, R=-1, Pabs=-1, Qabs=-1, Rabs=-1,
      A, B, Aabs, Babs, i, i1, duplicate, refl=0, tmp,
      *pnum, j, j1, n, k, k1, freepos, m, mabs,
      lowest, highest;
  double x, y, z, a, b, c, d, h,
      xP, xQ, xR, yP, yQ, yR, zP, zQ, zR,
      len, fact;
  float alpha=0.0, Sx=1.0, Sy=1.0, Sz=1.0,
      xC, yC, zC, C1, C2, C3,
      xlowest=1.0, xhighest, x1=0.0, y1=0.0, z1=0.0;
  while (nmax > 0 && p[nmax-1].inuse == 0) nmax--;
  duplicate = (query(LIN2, "'Move'/'Copy'? (M/C) ") == 'C');
  enquire(LIN2, "Ondergrens: ", &xlowest);
  lowest = (int)(xlowest + 0.1);
  if (lowest < 1) lowest = 1;
  xhighest = nmax - 1;
  enquire(LIN3, "Bovengrens: ", &xhighest);
  highest = (int)(xhighest + 0.1);
  if (highest >= nmax) highest = nmax - 1;
  if (duplicate)
  { freepos = nmax;
    check_alloc(nmax + highest - lowest);
  }
  grmes(LIN3, " ");
  ch = query(LIN2, "Rotatie? (J/N) ");
  if (ch == 'J')
  { grmes(LIN1, "Rotatie om PQ.");
    grmes(LIN2, "Puntnrs. P and Q:");
    bufposition = -1;
    if (!rdoldnr(&P) || !rdoldnr(&Q)) return;
    Pabs = abs(P); Qabs = abs(Q);
    x = p[Pabs].xw; y = p[Pabs].yw; z = p[Pabs].zw;
    x1 = p[Qabs].xw; y1 = p[Qabs].yw; z1 = p[Qabs].zw;
    enquire(LIN2, "Hoek in graden:", &alpha);
    alpha = alpha * PIdiv180;
    initrotate(x, y, z, x1 - x, y1 - y, z1 - z, alpha);
    for (i=lowest; i<=highest; i++)
    if (p[i].inuse && (P > 0 || i != Pabs) &&
                     (Q > 0 || i != Qabs))
    { rotate(p[i].xw, p[i].yw, p[i].zw, &x, &y, &z);
      if (duplicate)
      { i1 = freepos++; p[i1].inuse = 1; p[i].inuse = i1;
      } else i1 = i;
      p[i1].xw = x; p[i1].yw = y; p[i1].zw = z;
    }
```

```
) else
{ ch = query(LIN2, "Translatie? (J/N) ");
  if (ch == 'J')
  { grmes(LIN2, " ");
    if (query(LIN1, "Schuifvector? (J/N) ") != 'J')
    { enquire(LIN1, "delta x = ", &x1);
      enquire(LIN2, "delta y = ", &y1);
      enquire(LIN3, "delta z = ", &z1);
    } else
    { grmes(LIN1, "AB is schuifvector");
      grmes(LIN2, "Puntnrs. A en B:");
      bufposition = -1;
      if (!rdoldnr(&A) || !rdoldnr(&B)) return;
      Aabs = abs(A); Babs = abs(B);
      x = p[Aabs].xw; y = p[Aabs].yw; z = p[Aabs].zw;
      x1 = p[Babs].xw - x;
      y1 = p[Babs].yw - y;
      z1 = p[Babs].zw - z;
    }
    for (i=lowest; i<=highest; i++)
    if (p[i].inuse &&
       (A > 0 || i != Aabs) && (B > 0 || i != Babs))
    { if (duplicate)
      { i1 = freepos++; p[i1].inuse = 1; p[i].inuse = i1;
      } else i1 = i;
      p[i1].xw = p[i].xw + x1;
      p[i1].yw = p[i].yw + y1;
      p[i1].zw = p[i].zw + z1;
    }
  } else
  { ch = query(LIN2, "Schaling? (J/N) ");
    if (ch == 'J')
    { if (query(LIN2, "Uniform? (J/N) ") == 'J')
      { enquire(LIN3, "Sx=Sy=Sz= ", &Sx); Sy = Sz = Sx;
      } else
      { grmes(LIN2, " ");
        enquire(LIN1, "Sx = ", &Sx);
        enquire(LIN2, "Sy = ", &Sy);
        enquire(LIN3, "Sz = ", &Sz);
      }
      while (1)
      { grmes(LIN1, "Vast punt:");
        grmes(LIN2, "Centr.(C), Oorspr.(O)");
        ch = query(LIN3, "of een Hoekpunt(H):");
        if (ch == 'C')
        { checkall(); xC = x0; yC = y0; zC = z0;
        } else
        if (ch == 'O' || ch == '0')
```

```
        xC = yC = zC = 0.0; else
      if (ch == 'H')
      { grmes(LIN2, "Puntnummer: ");
        bufposition = -1;
        if (!rdoldnr(&P)) return;
        Pabs = abs(P);
        xC = p[Pabs].xw;
        yC = p[Pabs].yw;
        zC = p[Pabs].zw;
      } else continue;
      break;
    }
    C1 = xC * (1.0 - Sx);
    C2 = yC * (1.0 - Sy);
    C3 = zC * (1.0 - Sz);
    for (i=lowest; i<=highest; i++)
    if (p[i].inuse && (P > 0 || i != Pabs))
    { if (duplicate)
      { i1 = freepos++;
        p[i1].inuse = 1; p[i].inuse = i1;
      } else i1 = i;
      p[i1].xw = Sx * p[i].xw + C1;
      p[i1].yw = Sy * p[i].yw + C2;
      p[i1].zw = Sz * p[i].zw + C3;
    }
  } else
  { ch = query(LIN2, "Spiegeling? (J/N) ");
    if (ch == 'J')
    { grmes(LIN1, "PQR is spiegelvlak.");
      grmes(LIN2, "Puntnrs. P, Q, R:");
      bufposition = -1; refl = 1;
      if (!rdoldnr(&P) || !rdoldnr(&Q) || !rdoldnr(&R))
        return;
      Pabs = abs(P); Qabs = abs(Q); Rabs = abs(R);
      xP = p[Pabs].xw; yP = p[Pabs].yw; zP = p[Pabs].zw;
      xQ = p[Qabs].xw; yQ = p[Qabs].yw; zQ = p[Qabs].zw;
      xR = p[Rabs].xw; yR = p[Rabs].yw; zR = p[Rabs].zw;
      a = yP*zQ + zP*yR + yQ*zR - yP*zR - zP*yQ - zQ*yR;
      b = xP*zQ + zP*xR + xQ*zR - xP*zR - zP*xQ - zQ*xR;
      c = xP*yQ + yP*xR + xQ*yR - xP*yR - yP*xQ - yQ*xR;
      len = sqrt(a*a+b*b+c*c);
      if (len == 0.0) len = 1e-15;
      a /= len; b /= len; c /= len;
      d = a*xP + b*yP + c * zP;
      for (i=lowest; i<=highest; i++)
      if (p[i].inuse &&
      (P > 0 || i != Pabs) &&
      (Q > 0 || i != Qabs) &&
```

```
        (R > 0 || i != Rabs))
        { if (duplicate)
          { i1 = freepos++;
            p[i1].inuse = 1; p[i].inuse = i1;
          } else i1 = i;
          x = p[i].xw; y = p[i].yw; z = p[i].zw;
          h = a*x +b*y + c*z; fact = 2.0 * (d-h);
          p[i1].xw = x + fact * a;
          p[i1].yw = y + fact * b;
          p[i1].zw = z + fact * c;
        }
      } else {grmes(LIN2, " "); return;}
    }
  }
}
if (duplicate)
{ grmes(LIN1, "Nieuwe punten:");
  sprintf(str, "%d-%d", nmax, freepos-1);
  pressanykey(str);
  nmax = freepos;
}
grmes(LIN3, "Wacht s.v.p. ...");

modified = 1; checkall();
while (nface >0 && pface[nface-1] == NULL) nface--;
if (duplicate)
{ if (2*nface >= MAXFACES)
  { grmes(LIN3, "Te veel grensvlakken"); getch(); display();
    return;
  }
  freepos = nface;
  for (j=0; j<nface; j++)
  { if (pface[j] == NULL) continue;
    j1 = freepos++;
    pface[j1] = NULL;
    n = pface[j][0];
    if (n<0) ermes("n<0 in transform");
    if ((pface[j1] = (int *)farcalloc(n+1, sizeof(int)))
       == NULL)
      ermes("Niet genoeg geheugen");
    pface[j1][0] = n;
    for (k=1; k<=n; k++)
    { k1 = (refl ? (n > 3 ? (k < 4 ? 4-k : 4+n-k)
                          : n+1-k)
                 : k);
      m = pface[j][k1]; mabs = abs(m);
      if (mabs < lowest || mabs > highest) break;
      pface[j1][k] =
```

```
        (mabs == Pabs && P < 0 ||
         mabs == Qabs && Q < 0 ||
         mabs == Rabs && R < 0
         ? m : (m < 0 ? -p[mabs].inuse : p[m].inuse));
      }
      if (mabs < lowest || mabs > highest)
      { farfree(pface[j1]); freepos--;
      }
    }
    nface = freepos;
  } else
  if (refl)   /* Keer de orientatie van grensvlakken om; */
  { for (j=0; j<nface; j++)
    { if (pface[j] == NULL) continue;
      n = pface[j][0];
      if (n < 3) continue;
      pnum = pface[j];
      tmp = pnum[1]; pnum[1] = pnum[3]; pnum[3] = tmp;
      k1 = (n - 3)/2;
      for (k=1; k<=k1; k++)
      { tmp = pnum[3+k];
        pnum[3+k] = pnum[n+1-k];
        pnum[n+1-k] = tmp;
      } /* E.g. old: 1 2 3 4 5 6 7 8. */
    }   /*      new: 3 2 1 8 7 6 5 4. */
  }     /* (Elk hoekpunt behalve 2 mag concaaf zijn) */
  display();
}

static void viewing(float x, float y, float z,
                    double *pxe, double *pye, double *pze)
{ *pxe = v11*x + v21*y;
  *pye = v12*x + v22*y + v32*z;
  *pze = v13*x + v23*y + v33*z + v43;
  if (*pze <= EPS)
    ermes("Gebruik s.v.p. een grotere waarde voor rho");
}

static void wrfile(void)
{ int i, j, n, k;
  char ch;
  FILE *fp;
  grmes(LIN2, "Uitvoerfile: ");
  if (kbhit()) getch();
  if (*filnam) grmes(LIN3, filnam);
  textcursor(margin + 8*strlen(filnam)+8, LIN3);
  if (ch = mygetch(), ch != '\n' && ch != '\r')
  { ungetch(ch);
```

```
    getstr(margin, LIN3, filnam);
  }
  if (*filnam == '\0' || (fp = fopen(filnam, "w")) == NULL)
  { grmes(LIN3, "Kan file niet openen"); return;
  }
  for (i=0; i<nmax; i++)
  if (p[i].inuse)
    fprintf(fp, "%d %f %f %f\n",
                i, p[i].xw, p[i].yw, p[i].zw);
  if (nface > 0)
  { fprintf(fp, "Faces:\n");
    for (j=0; j<nface; j++)
    { if (pface[j] == NULL) continue;
      n = pface[j][0];
      if (n < 0) ermes("n<0 in wrfile");
      for (k=1; k<=n; k++) fprintf(fp, " %d", pface[j][k]);
      fprintf(fp, ".\n");
    }
  }
  grmes(LIN2, " ");
  grmes(LIN3, " ");
  fclose(fp);
  modified = 0; /* File en datastructuur identiek */
}

static void zoom(int code)
/* 'zoom' kan na 'initgr' worden aangeroepen */
{ if (code)  /*  1 = large;  0 = small  */
  { Xvp_min = 0.25; Xvp_max = x_max - 0.25;
    Yvp_min = 0.4; Yvp_max = y_max - 0.4;
  } else
  { Xvp_min = 0.25; Xvp_max = 6.3;
    /* Om ruimte voor commando's te reserveren */
    Yvp_min = 0.4; Yvp_max = 6.7;
        /* zoom(0) zal niet worden aangeroepen als pridim = 1 */
        /* dus we hebben x_max = 10.0, y_max = 7.0           */
  }
  zoomin = code;
  Xvp_range = Xvp_max - Xvp_min;
  Yvp_range = Yvp_max - Yvp_min;
  Xvp_center = 0.5*(Xvp_min + Xvp_max);
  Yvp_center = 0.5*(Yvp_min + Yvp_max);
}
```

APPENDIX B:
PROGRAMMATEKST VAN MODULE HLPFUN

```
/* HLPFUN: Een functie voor de eliminatie van verborgen
           lijnen, die gebruik maakt van een verdeling
           van het scherm in rechthoeken.
           Deze versie bevat een voorziening voor het
           werken met gekromde oppervlakken.
*/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include "grpack.h"
#define max(x,y)    ((x)>(y)?(x):(y))
#define min(x,y)    ((x)<(y)?(x):(y))

#define max3(x,y,z) ((x)>(y)?max(x,z):max(y,z))
#define min3(x,y,z) ((x)<(y)?min(x,z):min(y,z))
#define xwhole(x)   ((int)(((x)-xmin)/deltaX))
#define ywhole(y)   ((int)(((y)-ymin)/deltaY))
#define xreal(i)    (xmin+((i))*deltaX)

#define M -1000000.0
#define NSCREEN 15
#define BIG 1.e30
int abs(int i);

static void
  check_kbhit(void),
  add_linesegment(int P, int Q),
  viewing(double x, double y, double z,
    double *pxe, double *pye, double *pze),
  linesegment(double xP, double yP, double zP,
            double xQ, double yQ, double zQ, int k);

void
```

```
  ermes(char *str), /* See also module D3D */
  coeff(float rho, float theta, float phi),
  init_viewport(void);

static int
  counter_clock(int i0, int i1, int i2, int code),
  includesvertex(int h, int i, int j),
  sameside(double xj, double yj,
           double xl, double yl,
           double xh, double yh,
           double xi, double yi),
  allocsize(int divfactor, int elsize);

extern int nmax, nface, **pface;
extern float x0, y0, z0, xmin, xmax, ymin, ymax;

static int ipixmin, ipixmax, ipixleft, ipixright,
  ipix, jpix, jtop, jbot, j_old, jI, topcode[3], *POLY,
  npolyalloc, *vertexconvex, ntrsetalloc, npoly,
  isize=sizeof(int), LOWER[NSCREEN], UPPER[NSCREEN],
  LOW[NSCREEN], UP[NSCREEN], ntrset, maxvertex, jface,
  nTRIANGLE, nTRLIST, iTRIANGLE, iTRLIST, inode, itria,
  *trset;



extern double v11, v12, v13, v21, v22, v23, v32, v33, v43,
  PIdiv180;
static double d, c1, c2,
  eps=1e-5, meps=-1e-5, oneplus=1+1.e-5,
  Xrange, Yrange, Xvp_range, Yvp_range,
  deltaX, deltaY, denom, slope,
  Xleft[3], Xright[3], Yleft[3], Yright[3],
  a, b, c, surfl;

struct linsegface
{ int i; double a, b, c; struct linsegface *next;
} *lsegfacenode(void);

struct vertex
{ int inuse;
  float xw, yw, zw;
  double xe, ye, ze;
  struct linsegface *connect;
} *VERTEX, *pvertex;

extern struct vertex *p;
```

```
static struct triangle
{ int A, B, C;
  double a, b, c, h;
} *TRIANGLE, *ptriangle;

static struct lseglist
{ double xP, yP, zP, xQ, yQ, zQ;
  int k;
  struct lseglist *next;
} *lsegstart, *auxlseg, *lsegnode(void);


struct trlist { unsigned int ptria; unsigned int next; }
  *TRLIST, *pnode;

struct
{ int tr_cov;
  float tr_dist;
  unsigned int start;
} SCREEN[NSCREEN][NSCREEN], *pointer;



void hlpfun(float rho, float theta, float phi,
            float surflimit)
{ int P, Q, ii, vertexnr, i, kk, i0, i1, i2, code, count,
  polygonconvex, loopcount, jtr, j, n, k;

  struct linsegface *ptr, *dummy, *dummy0;

  double
    Xvp_min=0, Xvp_max, Yvp_min=0, Yvp_max,
    fx, fy, Xcentre, Ycentre, Xvp_centre, Yvp_centre,
    xP, yP, zP, xQ, yQ, zQ, XP, YP, XQ, YQ,
    Xlft, Xrght, Ylft, Yrght,
    xxP, yyP, zzP, xxQ, yyQ, zzQ;

 textXY(40, 40, "Wacht s.v.p. ...");
 Xvp_max = x_max; Yvp_max = y_max;
 surfl = surflimit;
 npolyalloc=200;
 POLY=(int *)farcalloc((long)npolyalloc, (long)isize);
 vertexconvex=
   (int *)farcalloc((long)npolyalloc, (long)isize);
 if (POLY == NULL || vertexconvex == NULL)
 { ermes("Geheugenprobleem: veelhoek");
 }
```

```
VERTEX = p;
  /* Nu wijst VERTEX naar hetzelfde geheugengebied als
     p in module D3D
  */

/* Initialize screen matrix */
for (ipix=0; ipix<NSCREEN; ipix++)
for (jpix=0; jpix<NSCREEN; jpix++)
{ pointer=&(SCREEN[ipix][jpix]);
  pointer->tr_cov=-1; pointer->tr_dist=BIG;
  pointer->start=-1;
}

coeff(rho, theta, phi);
maxvertex = nmax-1; /* nmax wordt gedefinieerd in module D3D */
dummy = lsegfacenode();
for (i=1; i<=maxvertex; i++)
  VERTEX[i].connect = (VERTEX[i].inuse ? NULL : dummy);
        /* dummy: niet in gebruik              */
        /* NULL: lege lijst (maar in gebruik)  */

/* Bereken schermconstanten */
Xrange=xmax-xmin; Yrange=ymax-ymin;
Xvp_range=Xvp_max-Xvp_min; Yvp_range=Yvp_max-Yvp_min;
fx=Xvp_range/Xrange; fy=Yvp_range/Yrange;
d=(fx<fy ? fx : fy);
Xcentre=0.5*(xmin+xmax); Ycentre=0.5*(ymin+ymax);
Xvp_centre=0.5*(Xvp_min+Xvp_max);
Yvp_centre=0.5*(Yvp_min+Yvp_max);
c1=Xvp_centre-d*Xcentre; c2=Yvp_centre-d*Ycentre;
deltaX=oneplus*Xrange/NSCREEN;
deltaY=oneplus*Yrange/NSCREEN;
/* Now we have:  Xrange/deltaX < NSCREEN */

iTRLIST = 0; iTRIANGLE = 0;
nTRLIST = allocsize(4, sizeof(struct trlist));
TRLIST = (struct trlist *)farcalloc((long)nTRLIST,
 (long)sizeof(struct trlist));

nTRIANGLE = allocsize(3, sizeof(struct triangle));
TRIANGLE = (struct triangle *)farcalloc((long)nTRIANGLE,
 (long)sizeof(struct triangle));

if (TRLIST == NULL || TRIANGLE == NULL)
  ermes("Geheugenprobleem: driehoeken");

for (j=0; j<nface; j++)
{ if (pface[j] == NULL) continue;
```

```
n = pface[j][0];
if (n < 0) ermes("Interne fout: n < 0");
if (n == 1) ermes("Veelhoek met maar 1 hoekpunt");
POLY[0] = pface[j][1]; npoly = 1;
for (k=2; k<=n; k++)
{ i = pface[j][k];
  if (npoly==npolyalloc)
  { npolyalloc += (npolyalloc>>2);
    POLY = (int *)
      farrealloc(POLY, (long)npolyalloc*isize);
    vertexconvex = (int *)
      farrealloc(vertexconvex, (long)npolyalloc*isize);
    if (POLY == NULL || vertexconvex == NULL)
      ermes("Geheugenprobleem: veelhoeken");
  }
  POLY[npoly++]=i;
}
if (npoly==1) ermes("Interne fout: npoly = 1");
if (npoly==2)
{ add_linesegment(POLY[0], POLY[1]); continue;
}
jface = j;
if (!counter_clock(0, 1, 2, 0)) continue;
                                        /* achtervlak */
for (i=1; i<=npoly; i++)
{ ii=i%npoly; code=POLY[ii]; vertexnr=abs(code);
  if (code<0) POLY[ii]=vertexnr; else
    add_linesegment(POLY[i-1], vertexnr);
}

/* Triangulatie van een veelhoek */

count=1; polygonconvex=0; i1=-1;
while (npoly>2)
{ if (!polygonconvex)
  { polygonconvex=1;
    for (ii=0; ii<npoly; ii++)
    { i0 = (ii==0 ? npoly-1 : ii-1);
      i2 = (ii==npoly-1 ? 0 : ii+1);
      vertexconvex[ii] = counter_clock(i0, ii, i2, 0);
      if (!vertexconvex[ii]) polygonconvex=0;
    }
  }
  loopcount=npoly;
  do
  { if (++i1 >= npoly) i1=0;
    i0 = (i1==0 ? npoly-1 : i1-1);
    i2 = (i1==npoly-1 ? 0 : i1+1);
```

```
      } while (--loopcount && !polygonconvex &&
        (!vertexconvex[i1] || includesvertex(i0, i1, i2)));
        /* Berg driehoek op: */
      counter_clock(i0, i1, i2, count++);
      npoly--;
      for (ii=i1; ii<npoly; ii++) POLY[ii]=POLY[ii+1];
    }
  }
  /* Voeg dichtstbijzijnde driehoeken toe aan schermlijsten */
  for (ipix=0; ipix<NSCREEN; ipix++)
  for (jpix=0; jpix<NSCREEN; jpix++)
  { pointer=&(SCREEN[ipix][jpix]);
    if (pointer->tr_cov >= 0)
    { pnode = TRLIST + iTRLIST;
      pnode->ptria = pointer->tr_cov;
      pnode->next  = pointer->start;
      pointer->start = iTRLIST;
      if (++iTRLIST == nTRLIST)
        ermes("Geheugenprobleem: driehoekenlijst");
    }
  }
  farfree(POLY); farfree(vertexconvex);
  ntrsetalloc = 1000;
  trset = (int *)
    farcalloc(ntrsetalloc, sizeof(unsigned int));
  if (trset == NULL) ermes("Geheugenprobleem: verz. driehoeken");
  lsegstart = NULL;
  clearpage();
  init_viewport();

  /* Teken alle lijnstukken voor zover ze zichtbaar zijn */
  for (P=1; P<=maxvertex; P++)
  { pvertex=VERTEX+P; /* = &VERTEX[P] */
    ptr = pvertex->connect;
    if (ptr == dummy || ptr == NULL) continue;
    xP = pvertex->xe; yP = pvertex->ye; zP = pvertex->ze;
    XP= xP/zP; YP=yP/zP;
    for ( ; ptr != NULL; ptr = ptr->next)
    { Q = ptr->i;
      pvertex=VERTEX+Q; /* = &VERTEX[Q] */
      xQ = pvertex->xe; yQ = pvertex->ye; zQ = pvertex->ze;
      XQ=xQ/zQ; YQ=yQ/zQ;
      /* Gebruik makend van de schermlijsten, gaan we de
         verzameling driehoeken opbouwen die punten van
         PQ kunnen verbergen.
      */
      if (XP<XQ || (XP==XQ && YP<YQ))
      {Xlft=XP; Ylft=YP; Xrght=XQ; Yrght=YQ; } else
```

```
{Xlft=XQ; Ylft=YQ; Xrght=XP; Yrght=YP; }
ipixleft=xwhole(Xlft); ipixright=xwhole(Xrght);
denom=Xrght-Xlft;
if (ipixleft != ipixright) slope=(Yrght-Ylft)/denom;
jbot=jtop=ywhole(Ylft);
for (ipix=ipixleft; ipix<=ipixright; ipix++)
{ if (ipix==ipixright) jI=ywhole(Yrght); else
       jI=ywhole(Ylft+(xreal(ipix+1)-Xlft)*slope);
  LOWER[ipix]=min(jbot,jI); jbot=jI;
  UPPER[ipix]=max(jtop,jI); jtop=jI;
}
ntrset=0;
for (ipix=ipixleft; ipix<=ipixright; ipix++)
for (jpix=LOWER[ipix]; jpix<=UPPER[ipix]; jpix++)
{ pointer=&(SCREEN[ipix][jpix]);
  inode= pointer->start;
  while (inode >= 0)
  /* At the end of the list we have inode=-1 */
  { pnode = TRLIST + inode; itria = pnode->ptria;
    /* De driehoek zal alleen worden opgeborgen als hij
       nog niet in array trset aanwezig is.
    */
    if (ntrset==ntrsetalloc)
    { ntrsetalloc += (ntrsetalloc >> 2);
      trset = (int *)
       farrealloc(
       trset, (long)ntrsetalloc*sizeof(unsigned int));
      if (trset == NULL)
        ermes("Geheugenprobleem: verzameling driehoeken");
    }
    trset[ntrset]=itria; /* sentinel */
    jtr=0;
    while (trset[jtr]!=itria) jtr++;
    if (jtr==ntrset)
    { ntrset++; /* Store triangle */
    }
    inode=pnode->next;
  }
}
/* Nu is trset[0], ..., trset[ntrset-1] de verzameling */
/* driehoeken die misschien punten van PQ verbergen.   */
linesegment(xP, yP, zP, xQ, yQ, zQ, 0);
while (lsegstart != NULL)
{ xxP=lsegstart->xP;
  yyP=lsegstart->yP;
  zzP=lsegstart->zP;
  xxQ=lsegstart->xQ;
  yyQ=lsegstart->yQ;
```

```
        zzQ=lsegstart->zQ;
        kk=lsegstart->k;
        auxlseg = lsegstart; lsegstart = lsegstart->next;
        farfree(auxlseg);
        linesegment(xxP, yyP, zzP, xxQ, yyQ, zzQ, kk);
      }
    }
  }
  farfree(dummy); farfree(trset);
  for (i=1; i<=maxvertex; i++)
  if (VERTEX[i].inuse)
  { dummy = VERTEX[i].connect;
    while (dummy != NULL)
    { dummy0 = dummy;
      dummy = dummy->next;
      farfree(dummy0);
    }
  }
  farfree(TRLIST);
  farfree(TRIANGLE);
}


void coeff(float rho, float theta, float phi)
{ double th, ph, costh, sinth, cosph, sinph;
  /* Angles in radians: */
  th=theta*PIdiv180; ph=phi*PIdiv180;
  costh=cos(th); sinth=sin(th);
  cosph=cos(ph); sinph=sin(ph);
  /* Elements of viewing matrix V: */
  v11=-sinth; v12=-cosph*costh; v13=-sinph*costh;
  v21=costh;  v22=-cosph*sinth; v23=-sinph*sinth;
              v32=sinph;        v33=-cosph;
                                v43=rho;
}


static void add_linesegment(int P, int Q)
/* Deze functie kan lijnstuk PQ plaatsen in de lineaire lijst
   die begint in VERTEX[P].connect. (Indien nodig verwisselen
   we eerst de waarden van P en Q zodat P de kleinste van de
   twee wordt.) Als PQ al eerder in de lijst is opgeborgen, dan
   wordt het niet gedupliceerd. Als een eerder opgeborgen lijn-
   stuk PQ wordt gevonden, dan vergelijken we het grensvlak
   waartoe het behoort met het grensvlak waartoe het nieuwe
   lijnstuk behoort. Als de normaalvectoren van deze grensvlakken
   bijna evenwijdig zijn (als het inwendig produkt van hun nor-
   maalvectoren groter is dan 'surfl', dan wordt het oude
```

```
    lijnstuk verwijderd.
*/
{ struct linsegface *ptr, *new, **pp, **pp0;
  int iaux;
  if (P>Q) { iaux=P; P=Q; Q=iaux; }
  /* Now: P < Q */
  pp0 = pp = &(VERTEX[P].connect);
  ptr = *pp;
  while (ptr != NULL && ptr->i != Q)
  { pp = &(ptr->next); ptr = *pp;
  }
  if (ptr == NULL)
  { new = lsegfacenode();
    new->i = Q; new->a = a; new->b = b; new->c = c;
    new->next = *pp0; *pp0 = new;
  } else
  if (a*(ptr->a) + b*(ptr->b) + c*(ptr->c) > surfl)
  { *pp = ptr->next; farfree(ptr);
  }
}


static int counter_clock(int i0, int i1, int i2, int code)
/* code = 0: Bereken orientatie:
   code = 1: Bereken a, b, c, h; berg de eerste driehoek op;
   code > 1: Controleer of volgende driehoek in hetzelfde
             vlak ligt; berg het op.
*/
{ int A=POLY[i0], B=POLY[i1], C=POLY[i2], i, l;
  double xA, yA, zA, xB, yB, zB, xC, yC, zC, r,
         XA, YA, XB, YB, XC, YC, h0,
         DA, DB, DC, D, DAB, DAC, DBC, aux, dist,
         xR, yR, dev, tol;
  static double h;

  if(A<0)A=-A; if(B<0)B=-B; if(C<0)C=-C;

  pvertex=VERTEX+A;
  xA = pvertex->xe; yA = pvertex->ye; zA = pvertex->ze;

  pvertex=VERTEX+B;
  xB = pvertex->xe; yB = pvertex->ye; zB = pvertex->ze;

  pvertex=VERTEX+C;
  xC = pvertex->xe; yC = pvertex->ye; zC = pvertex->ze;

  h0 = xA * (yB*zC - yC*zB) -
       xB * (yA*zC - yC*zA) +
```

```
    xC * (yA*zB - yB*zA);

if (code==0)
{ if (h0 <= eps) return 0;
   a = yA * (zB-zC) - yB * (zA-zC) + yC * (zA-zB);
   b = -(xA * (zB-zC) - xB * (zA-zC) + xC * (zA-zB));
   c = xA * (yB-yC) - xB * (yA-yC) + xC * (yA-yB);
   r = sqrt(a*a+b*b+c*c); if (r==0.0) r=eps;
   a = a/r; b = b/r; c = c/r; h = h0/r;
   return 1;
}
/* Als h0 = 0, dan gaat vlak ABC door E en verbergt niets.
   Als h0 < 0, dan is driehoek ABC een achtervlak.
   In beide gevallen wordt iTRIANGLE niet verhoogd en worden
   de driehoeken van de veelhoek niet opgeborgen.
   a, b en c zal gebruikt worden in add_linesegment.
*/
dev = fabs(a*xC+b*yC+c*zC-h); tol = eps+0.001*fabs(h);
if (dev > tol)
{ to_text();
  printf("Hoekpunten niet in hetzelfde vlak:\n");
  for (i=1; i<=pface[jface][0]; i++)
    printf("%d ", pface[jface][i]);
  exit(1);
}
ptriangle = TRIANGLE + iTRIANGLE;
ptriangle->A = A; ptriangle->B = B ; ptriangle->C = C;
ptriangle->a = a; ptriangle->b = b ; ptriangle->c = c;
ptriangle->h = h;
/* De driehoek wordt nu opgeborgen in de schermlijsten
   van de bijbehorende rechthoeken; eerst worden de
   arrays LOWER, UPPER, LOW en UP gedefinieerd:
*/
XA=xA/zA; YA=yA/zA;
XB=xB/zB; YB=yB/zB;
XC=xC/zC; YC=yC/zC;
DA=XB*YC-XC*YB; DB=XC*YA-XA*YC; DC=XA*YB-XB*YA;
D=DA+DB+DC;
DAB=DC-M*(XA-XB); DAC=DB-M*(XC-XA); DBC=DA-M*(XB-XC);
topcode[0]=(D*DAB>0); topcode[1]=(D*DAC>0);
topcode[2]=(D*DBC>0);
Xleft[0]=XA; Yleft[0]=YA; Xright[0]=XB; Yright[0]=YB;
Xleft[1]=XA; Yleft[1]=YA; Xright[1]=XC; Yright[1]=YC;
Xleft[2]=XB; Yleft[2]=YB; Xright[2]=XC; Yright[2]=YC;
for (l=0; l<3; l++)  /* l = triangle-side number */
if (Xleft[l]>Xright[l] ||
   (Xleft[l]==Xright[l] && Yleft[l]>Yright[l]))
{ aux=Xleft[l]; Xleft[l]=Xright[l]; Xright[l]=aux;
```

```
  aux=Yleft[l]; Yleft[l]=Yright[l]; Yright[l]=aux;
}
ipixmin=xwhole(min3(XA,XB,XC));
ipixmax=xwhole(max3(XA,XB,XC));
for (ipix=ipixmin; ipix<=ipixmax; ipix++)
{ LOWER[ipix]=UP[ipix]=10000;
  UPPER[ipix]=LOW[ipix]=-10000;
}

for (l=0; l<3; l++)
{ ipixleft=xwhole(Xleft[l]); ipixright=xwhole(Xright[l]);
  denom=Xright[l]-Xleft[l];
  if (ipixleft != ipixright)
  slope=(Yright[l]-Yleft[l])/denom;
  j_old=ywhole(Yleft[l]);
  for (ipix=ipixleft; ipix<=ipixright; ipix++)
  { if (ipix==ipixright) jI=ywhole(Yright[l]); else
       jI=ywhole(Yleft[l]+(xreal(ipix+1)-Xleft[l])*slope);
    if (topcode[l])
    { UPPER[ipix]=max3(j_old,jI,UPPER[ipix]);
      UP[ipix]=min3(j_old,jI,UP[ipix]);
    } else
    { LOWER[ipix]=min3(j_old,jI,LOWER[ipix]);
      LOW[ipix]=max3(j_old,jI,LOW[ipix]);
    }
    j_old=jI;
  }
}
/* Voor schermkolom ipix wordt de driehoek alleen met
   rechthoeken in de rijen LOWER[ipix], ..., UPPER[ipix]
   geassocieerd. Het deel LOW[ipix]+1, ..., UP[ipix]-1
   van deze rijen betreffen rechthoeken die volledig
   binnen de driehoek liggen.
*/
for (ipix=ipixmin; ipix<=ipixmax; ipix++)
for (jpix=LOWER[ipix]; jpix<=UPPER[ipix]; jpix++)
{ pointer=&(SCREEN[ipix][jpix]);
  if (jpix>LOW[ipix] && jpix<UP[ipix])
  { xR=xmin+(ipix+0.5)*deltaX;
    yR=ymin+(jpix+0.5)*deltaY;
    denom=a*xR+b*yR+c*d;
    dist =
    fabs(denom)>eps ? h*sqrt(xR*xR+yR*yR+1.)/denom : BIG;
    /* De lijn vanaf oogpunt E naar punt (xR, yR, 1) in
       het midden van de rechthoek snijdt vlak ABC op een
       afstand 'dist' van E.
    */
    if (dist < pointer->tr_dist)
```

```
      { pointer->tr_cov=iTRIANGLE; pointer->tr_dist=dist;
      }
    } else /* Voeg driehoek toe aan schermlijst */
    { pnode = TRLIST + iTRLIST;
      pnode->ptria = iTRIANGLE;
      pnode->next = pointer->start;
      pointer->start = iTRLIST;
      if (++iTRLIST == nTRLIST)
       ermes("Memory: Triangle list");
    }
  }
  if (++iTRIANGLE == nTRIANGLE)
    ermes("Geheugenprobleem: driehoeken");
  return 1;
}


static void viewing(double x, double y, double z,
                    double *pxe, double *pye, double *pze)
{ *pxe = v11*x + v21*y;
  *pye = v12*x + v22*y + v32*z;
  *pze = v13*x + v23*y + v33*z + v43;
}


static void linesegment(double xP, double yP, double zP,
                   double xQ, double yQ, double zQ, int k)
{ /* Lijnstuk PQ moet worden getekend, voor zover het niet
     verborgen wordt door de driehoeken trset[k] t/m
     trset[ntrset-1].
  */
  int A, B, C, i, Pbeyond, Qbeyond,
    outside, Poutside, Qoutside, eA, eB, eC, sum;
  double a, b, c, h, hP, hQ, r1, r2, r3,
    xA, yA, zA, xB, yB, zB, xC, yC, zC,
    dA, dB, dC, labmin, labmax, lab, mu,
    xmin, ymin, zmin, xmax, ymax, zmax,
    C1, C2, C3, K1, K2, K3, denom1, denom2,
    Cpos, Ppos, Qpos, aux, eps1;
  struct triangle *ptriangle;


  while (k<ntrset)
  { itria=trset[k]; ptriangle=TRIANGLE+itria;
    a=ptriangle->a; b=ptriangle->b; c=ptriangle->c;
    h=ptriangle->h;

    /* Test 1: */
```

```
hP=a*xP+b*yP+c*zP; hQ=a*xQ+b*yQ+c*zQ;
eps1=eps+eps*h;
if (hP-h<=eps1 && hQ-h<=eps1) {k++; continue;}
                                    /* PQ niet achter ABC */
/* Test 2: */
K1=yP*zQ-yQ*zP; K2=zP*xQ-zQ*xP; K3=xP*yQ-xQ*yP;
A=ptriangle->A; B=ptriangle->B; C=ptriangle->C;
pvertex=VERTEX+A;
xA = pvertex->xe; yA = pvertex->ye; zA = pvertex->ze;
pvertex=VERTEX+B;
xB = pvertex->xe; yB = pvertex->ye; zB = pvertex->ze;
pvertex=VERTEX+C;
xC = pvertex->xe; yC = pvertex->ye; zC = pvertex->ze;
dA=K1*xA+K2*yA+K3*zA;
dB=K1*xB+K2*yB+K3*zB;
dC=K1*xC+K2*yC+K3*zC;
/* Als dA, dB en dC hetzelfde teken hebben liggen
   de hoekpunten A, B en C aan dezelfde kant van
   vlak EPQ.
*/
eA= dA>eps ? 1 : dA<meps ? -1 : 0;
eB= dB>eps ? 1 : dB<meps ? -1 : 0;
eC= dC>eps ? 1 : dC<meps ? -1 : 0;
sum = eA+eB+eC;
if (abs(sum)>=2) { k++; continue; }
/* >= 2: De (oneindige) lijn PQ ligt buiten piramide
         EABC (of de lijn en de piramide hebben ten
         hoogste 1 gemeenschappelijk punt).
   < 2:  Er is een snijpunt.
*/
  /* Test 3: */
Poutside=Qoutside=0; labmin=1.; labmax=0.;
for (i=0; i<3; i++)
{ C1=yA*zB-yB*zA; C2=zA*xB-zB*xA; C3=xA*yB-xB*yA;
  /* C1 x + C2 y + C3 z = 0  is vlak EAB */
  Cpos=C1*xC+C2*yC+C3*zC;
  Ppos=C1*xP+C2*yP+C3*zP;
  Qpos=C1*xQ+C2*yQ+C3*zQ;
  denom1=Qpos-Ppos;
  if (Cpos>eps)
  { Pbeyond= Ppos<meps; Qbeyond= Qpos<meps;
    outside= Pbeyond && Qpos<=eps ||
             Qbeyond && Ppos<=eps;
  } else if (Cpos<meps)
  { Pbeyond= Ppos>eps; Qbeyond= Qpos>eps;
    outside= Pbeyond && Qpos>=meps ||
             Qbeyond && Ppos>=meps;
  } else outside=1;
```

```
  if (outside) break;
  lab= fabs(denom1)<=eps  ? 1.e7 : -Ppos/denom1;
  /* lab indicates where PQ meets plane EAB */
  Poutside |= Pbeyond;
  Qoutside |= Qbeyond;
  denom2=dB-dA;
  mu= fabs(denom2)<=eps ? 1.e7 : -dA/denom2;
  /* mu tells where AB meets plane EPQ */
  if (mu>=meps && mu<=oneplus &&
      lab>=meps && lab<=oneplus)
  { if (lab<labmin) labmin=lab;
    if (lab>labmax) labmax=lab;
  }
  aux=xA; xA=xB; xB=xC; xC=aux;
  aux=yA; yA=yB; yB=yC; yC=aux;
  aux=zA; zA=zB; zB=zC; zC=aux;
  aux=dA; dA=dB; dB=dC; dC=aux;
}
if (outside) {k++; continue;}

/* Test 4: */
if (!(Poutside || Qoutside)) return; /* PQ onzichtbaar */

/* Test 5: */
r1=xQ-xP; r2=yQ-yP; r3=zQ-zP;
xmin=xP+labmin*r1; ymin=yP+labmin*r2;
zmin=zP+labmin*r3;
if (a*xmin+b*ymin+c*zmin-h<-eps1) { k++; continue; }
xmax=xP+labmax*r1; ymax=yP+labmax*r2;
zmax=zP+labmax*r3;
if (a*xmax+b*ymax+c*zmax-h<-eps1) { k++; continue; }
/* In dat geval ligt een snijpunt van PQ en de piramide
   dichterbij dan vlak ABC.
*/

/* Test 6: */
if (Poutside || hP<h-eps1)
{ auxlseg=lsegstart;
  lsegstart = lsegnode();
  lsegstart->xP = xP; lsegstart->yP = yP;
  lsegstart->zP = zP;
  lsegstart->xQ = xmin; lsegstart->yQ = ymin;
  lsegstart->zQ = zmin;
  lsegstart->k = k+1; lsegstart->next = auxlseg;
}
if (Qoutside || hQ<h-eps1)
{ auxlseg=lsegstart;
  lsegstart = lsegnode();
```

```
      lsegstart->xP = xmax; lsegstart->yP = ymax;
      lsegstart->zP = zmax;
      lsegstart->xQ = xQ; lsegstart->yQ = yQ;
      lsegstart->zQ = zQ;
      lsegstart->k = k+1; lsegstart->next = auxlseg;
    }
    return;
  }
    move(d*xP/zP+c1, d*yP/zP+c2);
  draw(d*xQ/zQ+c1, d*yQ/zQ+c2);
}


void init_viewport(void)
{ float len=0.3, len1=0.2;
  move(0.0, len); draw(0.0, 0.0); draw(len, 0.0);
  move(x_max-len, 0.0); draw(x_max, 0.0); draw(x_max, len);
  move(x_max, y_max-len1); draw(x_max, y_max);
  draw(x_max-len1, y_max);
  move(len1, y_max); draw(0.0, y_max);
  draw(0.0, y_max-len1);
  /* Het verschil tussen len en len1 stelt ons in staat
     boven- en onderkant van elkaar te onderscheiden.
  */
}


static int includesvertex(int h, int i, int j)
/* Ligt er een ander hoekpunt binnen de driehoek met
   hoekpunten h, i en j?
*/
{ int l;
  double zh, zi, zj, zl, XH, YH, XI, YI, XJ, YJ, XL, YL;

  pvertex = VERTEX + POLY[h]; zh = pvertex->ze;
  XH = pvertex->xe / zh; YH = pvertex->ye / zh;

  pvertex = VERTEX + POLY[i]; zi = pvertex->ze;
  XI = pvertex->xe / zi; YI = pvertex->ye / zi;

  pvertex = VERTEX + POLY[j]; zj = pvertex->ze;
  XJ = pvertex->xe / zj; YJ = pvertex->ye / zj;

  for (l=0; l<npoly; l++)
  { if (!vertexconvex[l] && l != h && l != i && l !=j)
    { pvertex = VERTEX + POLY[l];  zl = pvertex->ze;
      XL = pvertex->xe / zl;
      YL = pvertex->ye / zl;
```

```
      if (sameside(XH, YH, XL, YL, XI, YI, XJ, YJ) &&
          sameside(XI, YI, XL, YL, XJ, YJ, XH, YH) &&
          sameside(XJ, YJ, XL, YL, XH, YH, XI, YI))
        return 1;
    }
  }
  return 0;
}


static int sameside(double xj, double yj,
                    double xl, double yl,
                    double xh, double yh,
                    double xi, double yi)
/* Liggen de punten j en l aan dezelfde kant van lijn h-i?
*/
{ double a, b, c;
  a=yh-yi; b=xi-xh; c=xh*yi-yh*xi;
  return (a*xj+b*yj+c)*(a*xl+b*yl+c)>0.0;
}


struct lseglist *lsegnode(void)
{ struct lseglist *p;
  p = (struct lseglist *)
      farmalloc((long)sizeof(struct lseglist));
  if (p == NULL) ermes("Geheugenprobleem: PQ");
  return p;
}


struct linsegface *lsegfacenode(void)
{ struct linsegface *p;
  p = (struct linsegface *)
      farmalloc((long)sizeof(struct linsegface));
  if (p == NULL) ermes("Geheugenprobleem: ribben");
  return p;
}


static int allocsize(int divfactor, int elsize)
{ long lll;
  lll = (farcoreleft()/divfactor)/elsize;
  return (lll < 32767 ? (int)lll : 32767);
}
```

LITERATUUR

Ammeraal, L. (1988). *Turbo C*, Schoonhoven: Academic Service.

Ammeraal, L. (1986). *Programming Principles in Computer Graphics*, Chichester: John Wiley.

Ammeraal, L. (1986). *C for Programmers*, Chichester: John Wiley.

Ammeraal, L. (1987). *Computer Graphics for the IBM PC*, Chichester: John Wiley.

Ammeraal, L. (1987). *Programs and Data Structures in C*, Chichester: John Wiley.

Borland International (1987). *Turbo C User's Guide, Turbo C Reference Manual, Turbo C Version 1.5 Additions & Enhancements.*

Coxeter, H. S. M. (1961). *Introduction to Geometry*, New York: John Wiley.

Earle, J. H. (1987). *Engineering Design Graphics*, Reading, Mass.: Addison-Wesley.

Foley, J. D., and A. van Dam (1982). *Fundamentals of Interactive Computer Graphics*, Reading, Mass.: Addison-Wesley.

Hearn, D., and M. P. Baker (1986). *Computer Graphics*, Englewood Cliffs, NJ: Prentice-Hall.

Kernighan, B. W., and D. M. Ritchie (1978). *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall.

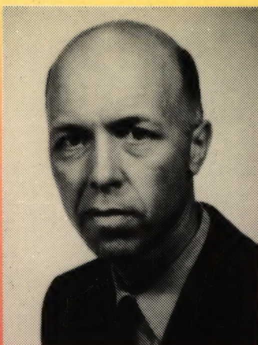
Knuth, D. H. (1968). *The Art of Computer Programming Vol. 1: Fundamental Algorithms*, Reading, Mass.: Addison-Wesley.

Newman, W. M., and R. F. Sproull (1979). *Principles of Interactive Computer Graphics*. New York: McGraw-Hill.

Norton, P. (1985). *Programmer's Guide to the IBM PC*, Washington: Microsoft Press.

Pal, I. (1974). *Raumgeometrie in der Technischen Praxis*, Budapest: Akademiai Kiado.

INDEX



Ir. L. Ammeraal is docent Informatica aan de Hogeschool Utrecht. Hij houdt zich vooral bezig met programmeren in C en met Computer Graphics en publiceerde - zowel in het Nederlands als in het Engels - over deze onderwerpen enige boeken, waarvan er sommige in andere talen zijn vertaald. Hij was in het verleden achtereenvolgens werkzaam bij het Dr. Neher Laboratorium van de PTT, bij Akzo Research & Engineering en bij de Stichting Mathematisch Centrum.

Interactieve 3D Computer Graphics bespreekt hoe perspectivische afbeeldingen van driedimensionale voorwerpen kunnen worden verkregen met behulp van een interactief programma, D3D geheten. Er is uitgegaan van een IBM PC (met HGA, CGA of EGA). De uitvoer kan geproduceerd worden op het beeldscherm, op een matrix-printer en desgewenst op een plotter. In D3D is een efficiënte algoritme voor de eliminatie van verborgen lijnen verwerkt; hiervan is gebruik gemaakt bij het vervaardigen van vele illustraties in dit boek. Naast D3D worden ook diverse hulpprogramma's besproken, zowel voor het genereren van meer gecompliceerde D3D-invoerfiles als voor 'deferred output' op een Hewlett-Packard plotter.

Het boek biedt meer dan een handleiding voor gebruikers. Het bespreekt ook de interne aspecten van de behandelde software, waarbij de aanpak van de geavanceerde onderwerpen 'solid modeling' en '3D Graphics' steeds praktisch en concreet is. De volledige programmatekst (in de taal C) is in het boek opgenomen.

Interactieve 3D Computer Graphics is geschreven voor een brede kring van lezers, namelijk voor professionele gebruikers en ontwikkelaars van CAD-software, voor docenten en studenten in technische vakken en informatica en voor hen die zich louter voor hun plezier met computer graphics bezighouden.

De behandelde programma's zijn ook op een tweetal diskettes bij de uitgever verkrijgbaar. Naast sourcecode - in Turbo C - bevatten deze diskettes ook de overeenkomstige .EXE-files, zodat gebruikers van een IBM PC de besproken programmatuur direct kunnen toepassen.

ISBN 90 6233 386 9
NUGI 856